



Attorney's Docket No. 07539.120

PATENT

**In The United States Patent And Trademark Office**

In re: Roy Hays  
Appl. No.: 09/782,685 Group Art Unit: 2155  
Filed: February 13, 2001 Examiner: Tran, Philip B.  
For: METHOD AND SYSTEM FOR COLLECTING INFORMATION AT  
DISTRIBUTED LOCATIONS

**DECLARATION OF BILLY W. HENSLEY AND ROY HAYS  
UNDER 37 C.F.R. §§ 1.131**

We, Billy W. Hensley and Roy Hay, being joint inventors of the claimed subject matter,  
do hereby declare and say as follows:

1. We are the joint inventors of the inventions claimed in the original and pending claims of the above-captioned patent application.
2. We have read and understand the above-captioned patent application, including the original specification and claims. We also have read and understand the Office Action dated March 6, 2006 and active claims 1-13.
3. We have read and understand the following art applied in the Office Action: U.S. Patent No. 6,692,436 to Bluth et al. (hereinafter "Bluth") filed on April 14, 2000.
4. We hereby incorporate by reference the CD-R filed on October 27, 2005. We submit that the creation date of each file contained on the CD-R and relied upon for overcoming the outstanding rejections predates the April 14, 2000 filing date of Bluth.
5. We hereby incorporate the report attached to my Declaration of November 22, 2004. The multiple entry dates subsequent to April 14, 2000 reflect dates on which files were

**BEST AVAILABLE COPY**

checked in or checked out from a database repository. Any modifications made to the files subsequent to April 14, 2000 were not necessary for or part of the first actual reduction to practice of the invention.

6. The software code set forth in the CD-R and existing prior to the April 14, 2000 filing date of Bluth constitutes an actual reduction to practice of the invention. We declare that the software code worked for its intended purpose and performed each and every function of claims 1-13.

7. We have attached a hardcopy print-out of the software code contained on the CD-R, as requested by the Examiner. The code has been annotated and line numbers added to assist the Examiner in identifying the relevant code for each claim feature. The top line of each page of the hardcopy print-out includes a directory path, with the file name corresponding to the "File(s)" set forth below in paragraphs 8-20. For example, for claim 1, phrase 1, the relevant code may be found at the hardcopy print-out with the path "C:\Documents and Settings\...\threadReceiver.cpp." at line 426 (on page 7 of the directory print-out).

8. With regard to claim 1, the following files existing prior to April 14, 2000 and submitted on the CD-R allowed claim 1 to be carried out prior to April 14, 2000:

*providing user information for registered users, the user information comprising medical information specific to the registered users;*

**Module:** LCKioskServer.exe

**File(s):** threadReceiver.h; threadReceiver.cpp

**Method(s):** CExportKiosks::buildDailyExport;

CExportLCUsers::buildDailyExport

**Line Nos.:** 426 *et seq.*

**Comments:** User information is packed into files and placed in FTP  
directory for kiosk pickup

*receiving updates to the user information;*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Method(s):** onExchange; exchangeFiles; pullFtpFiles

**Line Nos.:** 197 *et seq.*, 681 *et seq.*, 804 *et seq.*

**Comments:** Kiosks pick up data update files via FTP

*generating update user information;*

**Module:** LCKioskServer.exe

**File(s):** threadReceiver.h; threadReceiver.cpp

**Method(s):** CExportKiosks::buildDailyExport

CExportLCUsers::buildDailyExport

**Line Nos.:** 416 *et seq.*

**Comments:** Server generates files for kiosks containing user information.

The files are placed in a directory for pickup.

*for each of the collection kiosks, receiving a request from the collection kiosk for  
the generated update user information;*

**Module:** LCKioskClient.exe

**File(s):** threadReceiver.cpp; threadReceiver.h

**Method(s):** onTimerReceiveFiles

**Line Nos.:** 127 *et seq.*

**Comments:** All files are received via FTP. The LCKioskServer picks the

files up in the receiver directory and processes them all.

*sending to the requesting collection kiosk the update user information*

**Module:** LCKioskServer.exe

**File(s):** threadReceiver.cpp; threadReceiver.h

**Method(s):** CExportKiosks::buildDailyExport;  
CExportLCUsers::buildDailyExport

**Line Nos.:** 416 *et seq.*

**Comments:** Data from the server is stored on the kiosk in a local  
database. Logins on the kiosks are authenticated against  
database.

*storing the update user information at the requesting collection kiosk for  
subsequent requests,*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 265 *et seq.*

*wherein the collection kiosks use the update user information to verify whether a  
user is registered.*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 265 *et seq.*

9 With regard to claim 2, the following files existing prior to April 14, 2000  
allowed claim 2 to be carried out prior to April 14, 2000:

*wherein the collection kiosks operate as FTP clients and the computer system operates as an FTP server*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Method(s):** exchangeFiles; pullFtpFiles

**Line Nos.:** 681 *et seq.*, 804 *et seq.*

**Comments:** The server side FTP module is part of Windows operating system. The server generates files to be transferred and drops them into an FTP directory for kiosk pickup.

10. With regard to claim 3, the following files existing prior to April 14, 2000 allowed claim 3 to be carried out prior to April 14, 2000:

*wherein the received update user information includes indications of whether to add a registered user, delete a registered user, or change information relating to a registered user*

**Module (server side):** LCBroker.exe

**File(s):** xc\_applyKioskTrans.ccp

**Method(s):** applyUsers

**Line Nos.:** 93 *et seq.*

**Comments:** Determines if a user can be applied as a new user or must be rejected, i.e., if user is already in system

**Module (kiosk side):** KCData.dll

**File(s):** CoKCData.h; CoKCdata.ccp

**Method(s):** getLCUser

**Line Nos.:** 184 *et seq.*

**Comments:** User's information has been updated with information from the server after data exchange. The user is in one of several statuses: a verified lifeclinic user, a candidate to become a lifeclinic user, or rejected by server.

11. With regard to claim 4, the following files existing prior to April 14, 2000 allowed claim 4 to be carried out prior to April 14, 2000:

*wherein a collection kiosk sends a request for the generated update user information once a day*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Method(s):** onExchange; exchangeFiles; pullFtpFiles

**Line Nos.:** 197 *et seq.*, 681 *et seq.*, 804 *et seq.*

**Comments:** Kiosk pulls available updates

12. With regard to claim 5, the following files existing prior to April 14, 2000 allowed claim 5 to be carried out prior to April 14, 2000:

*wherein the user information includes a user identifier and a password*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Method(s):** onExchange; exchangeFiles; exportData

**Line Nos.:** 197 *et seq.*, 681 *et seq.*, 898 *et seq.*

**Module:** KCData.dll

**File(s):** CoKCData.h; CoKCData.cpp

**Method(s):** getUnexportedData

**Line Nos.:** 396 *et seq.*

**Comments:** Data sent to the server contains requests to add new users,  
with request including login and password.

13. With regard to claim 6, the following files existing prior to April 14, 2000  
allowed claim 6 to be carried out prior to April 14, 2000:

*providing user information for registered users, the user information comprising  
medical information specific to the registered users;*

**Module:** KCDData.dll

**File(s):** CoKCUser.h; CoKCUser.cpp

**Method(s):** get\_LCUser

**Line Nos.:** 184 *et seq.*

*sending a request for updated user information;*

**Module:** KioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Method(s):** onExchange; exchangeFiles

**Line Nos.:** 197 *et seq.*, 681 *et seq.*

*in response to sending the request, receiving the updated user information; and  
updating the provided user information for the registered user in accordance with the received  
updated user information so that the collection kiosk can verify whether a user of the collection  
kiosk is registered; and*

**Module:** KioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Method(s):** onExchange; exchangeFiles; pullFtpFiles

**Line Nos.:** 197 *et seq.*, 681 *et seq.*, 804 *et seq.*

**Module:** KCData.dll

**File(s):** CoKCData.h; CoKCData.cpp

**Method(s):** importLCUsers; getLCUser

**Line Nos.:** 627 *et seq.*, 184 *et seq.*

**Comments:** local user information updated with information from server  
via importLCUsers after data exchange.

*storing the updated user information at the collection kiosk for subsequent  
requests*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 265 *et seq.*

14. With regard to claim 7, the following files existing prior to April 14, 2000  
supported claim 7 prior to April 14, 2000:

*a central computer system for a web site, the central computer system providing a  
repository for the information, registering users of the web site, and accessing the information;  
and*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 265 *et seq.*

**Comments:** The central computer is comprised of IIS, MS SQL, FTP,  
LCKioskServer, and LLCBroker.



**Web Directories/Files:**

- i.     \Member\Admin\Kiosk
  - KioskCheck.asp; KioskSave.asp
- ii.    \Member\Admin\KioskAds
  - dataAdmin.asp; default.asp; distlist.asp; distlog.asp
  - distnew.asp; distsave.asp
- iii.   \Member\BloodPressure
  - BloodPressure.asp; BloodPressure\_4\_2.asp; LoadBP.asp;
  - LoadBP\_4\_2.asp; SaveBP.asp; SaveBP\_4\_2.asp
- iv.     \Member\Charts
  - Chart.asp; EmailtoPhysician.asp; NormalBPRanges.asp;
  - NormalCholesterolRanges.asp; Review.asp;
  - vitalchart.asp; NormalGlucoseRanges.asp
- v.      \Member>Login
  - index.asp; Login.asp; loginError.asp; verifyUser.asp
- vi.     \Member\MemberInfo
  - familyMember.asp; getSexCode.asp; LoadDependent.asp;
  - loadMemberInfo.asp; MemberInfo.asp;
  - MemberInfo\_new.asp; SaveDependent.asp;
  - saveMemberInfo.asp; saveMemberInfo\_3\_15.asp
- vii.    \Member\NewUser
  - agreement.asp; agrmdecline.asp; Newuser.asp;
  - Newuser\_form2.asp; Welcome.asp

viii. \Member\Preferences

- ChangePassword.asp; emailUpdatePreference.asp;  
LoadUserPreference.asp; mainPagePreference.asp;  
newsPreference.asp; password.asp; Preferences.asp;  
SaveEmailUpdate.asp; SaveMainPagePreference.asp;  
SaveNewsPreference.asp; SavePreference.asp

ix. \Member\Pulse

- LoadPulse.asp; LoadPulse\_4\_2.asp; Pulse.asp  
Pulse\_4\_2.asp; SavePulse.asp; SavePulse\_4\_2.asp

x. \Member\Weight

- LoadWeight.asp; LoadWeight\_4\_2.asp; SaveWeight.asp;  
SaveWEight\_4\_2.asp; Weight.asp; Weight\_4\_2.asp

*a plurality of collection kiosks for collecting information about users, for  
verifying whether a user is registered at the web site, and for sending the collected information  
to the central computer system when the user is registered.*

**Module:** KioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Method(s):** onExchange; exchangeFiles; pullFtpFiles

**Line Nos.:** 197 et seq., 681 et seq., 804 et seq.

**Comments:** Kiosks maintain local copy of centralized data

**Module:** KCData.dll

**File(s):** CoKCDATA.h; CoKCDATA.cpp

**Method(s):** getLCUser

**Line Nos.:** 184 *et seq.*

**Comments:** authenticates user against local database

15. With regard to claim 8, the following files existing prior to April 14, 2000  
allowed claim 8 to be carried out prior to April 14, 2000:

*wherein the information is medical information.*

**Module:** KCData.DLL

**File(s):** CoBPRreading.h; CoBPRreading.cpp;

**Line Nos.:** 104 *et seq.*,

**Module:** KCData.DLL

**File(s):** CoWeightReading.h; CoWeightReading.cpp;

**Line Nos.:** 78 *et seq.*

16. With regard to claim 9, the following files existing prior to April 14, 2000  
allowed claim 9 to be carried out prior to April 14, 2000:

*registering the users at the web site when information about a user is collected at  
one of a plurality of collection kiosks,*

**Module:** LCKioskServer.exe

**File(s):** threadReceiver.cpp; threadReceiver.h

**Method(s):** onTimerReceiveFiles

**Line Nos.:** 127 *et seq.*

**Comments:** all files received via FTP; LCKioskServer picks files up  
in the receive directory and processes through

LCBroker.exe

**Module:** LCBroker.exe

**File(s):** xc\_applyKioskTrans.cpp

**Method(s):** applyUsers

**Line Nos.:** 93 et seq.

*determining whether the user is registered at the website; and when registered, sending the collected information to a computer system so that the collected information is accessible to the user through the web site.*

**Module:** KCData.dll

**File(s):** CoKCUser.h; CoKCUser.cpp

**Method(s):** get\_LCUser

**Line Nos.:** 227 et seq.

**Module:** KioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Method(s):** onExchange; exchangeFiles

**Line Nos.:** 197 et seq., 681 et seq.

**Comments:** Each kiosk authenticates users against local database during

login. User statuses are verified user, candidate, and rejected.

Non-users and rejected users become candidate and registration is

scheduled to occur via subsequent data exchange with server.

17. With regard to claim 10, the following files existing prior to April 14, 2000 allowed claim 10 to be carried out prior to April 14, 2000:

*wherein a collection kiosk automatically sends a request for the generated update user information periodically.*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 197 *et seq.*

18. With regard to claim 11, the following files existing prior to April 14, 2000 allowed claim 11 to be carried out prior to April 14, 2000:

*wherein said sending a request for updated information is automatic and performed periodically.*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 235 *et seq.*

19. With regard to claim 12, the following files existing prior to April 14, 2000 allowed claim 12 to be carried out prior to April 14, 2000:

*wherein said sending a request for updated information is automatic and performed daily*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 123 *et seq.*

20. With regard to claim 13, the following files existing prior to April 14, 2000 allowed claim 13 to be carried out prior to April 14, 2000:

*the information comprising medical information specific to the registered users;*

**Module:** LCKioskClient.exe

**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 681 *et seq.*

*the central computer system further is for receiving updates to the user information from the collection kiosks, generating update user information, and for each of the collection kiosks, receiving a request from the collection kiosk for the generated update user information and sending to the requesting collection kiosk the update user information.*

**Module:** LCKioskClient.exe

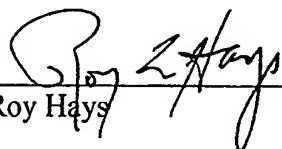
**File(s):** wndMonitorISP.h; wndMonitorISP.cpp

**Line Nos.:** 681 *et seq.*

21. We hereby declare that all statements made herein of our knowledge are true and all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the above-captioned patent application or any patent issued thereon.

\_\_\_\_\_  
Billy W. Hensley

\_\_\_\_\_  
Date

  
\_\_\_\_\_  
Roy Hays

25 Oct 2006  
Date

```
1 // CoKCUser.cpp : implementation of CoKCUser
2 #include "stdafx.h"
3 #include "KCDATA.h"
4 #include "CoKCUser.h"
5 #include "CoKCDATA.h"
6 #include "CoBPRReading.h"
7 #include "CoWeightReading.h"
8 #include "rs_kcdata.h"
9
10 //////////////////////////////////////
11 // CoKCUser
12
13 HRESULT CoKCUser::FinalConstruct()
14 {
15     m_pobjOwner = NULL;
16     m_prsBP = NULL;
17     m_prsWeight = NULL;
18
19     m_vKioskUserID.vt = VT_NULL;
20     m_vFirstName.vt = VT_NULL;
21     m_vLastName.vt = VT_NULL;
22     m_vMiddleName.vt = VT_NULL;
23     m_vAddress1.vt = VT_NULL;
24     m_vAddress2.vt = VT_NULL;
25     m_vCity.vt = VT_NULL;
26     m_vState.vt = VT_NULL;
27     m_vZip.vt = VT_NULL;
28     m_vPhone.vt = VT_NULL;
29     m_vPassword.vt = VT_NULL;
30     m_vEMail.vt = VT_NULL;
31     m_vLCPassword.vt = VT_NULL;
32     m_vLCUser.vt = VT_NULL;
33     m_vUserStatus.vt = VT_NULL;
34
35     return S_OK;
36 }
37
38 void CoKCUser::FinalRelease()
39 {
40     if (m_prsBP != NULL)
41         delete m_prsBP;
42
43     if (m_prsWeight != NULL)
44         delete m_prsWeight;
45
46     if (m_pobjOwner != NULL)
47         m_pobjOwner->Release();
48
49     return;
50 }
51
52 STDMETHODIMP CoKCUser::InterfaceSupportsErrorInfo(REFIID riid)
53 {
54     static const IID* arr[] =
55     {
56         IID_IKCUser
57     };
58     for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
59     {
60         if (InlineIsEqualGUID(*arr[i],riid))
61             return S_OK;
62     }
63     return S_FALSE;
64 }
65
66 STDMETHODIMP CoKCUser::get_KioskUserID(VARIANT *pVal)
```

```
67 {
68     *pVal = m_vKioskUserID;
69     return S_OK;
70 }
71
72 STDMETHODIMP CKCUser::get_FirstName(VARIANT * pVal)
73 {
74     VariantInit(pVal);
75     *pVal = _variant_t(m_vFirstName).Detach();
76     return S_OK;
77 }
78
79 STDMETHODIMP CKCUser::put_FirstName(VARIANT newVal)
80 {
81     m_vFirstName = newVal;
82     return S_OK;
83 }
84
85 STDMETHODIMP CKCUser::get_LastName(VARIANT *pVal)
86 {
87     VariantInit(pVal);
88     *pVal = _variant_t(m_vLastName).Detach();
89     return S_OK;
90 }
91
92 STDMETHODIMP CKCUser::put_LastName(VARIANT newVal)
93 {
94     m_vLastName = newVal;
95     return S_OK;
96 }
97
98 STDMETHODIMP CKCUser::get_MiddleName(VARIANT *pVal)
99 {
100     VariantInit(pVal);
101     *pVal = _variant_t(m_vMiddleName).Detach();
102     return S_OK;
103 }
104
105 STDMETHODIMP CKCUser::put_MiddleName(VARIANT newVal)
106 {
107     m_vMiddleName = newVal;
108     return S_OK;
109 }
110
111 STDMETHODIMP CKCUser::get_Address1(VARIANT *pVal)
112 {
113     VariantInit(pVal);
114     *pVal = _variant_t(m_vAddress1).Detach();
115     return S_OK;
116 }
117
118 STDMETHODIMP CKCUser::put_Address1(VARIANT newVal)
119 {
120     m_vAddress1 = newVal;
121     return S_OK;
122 }
123
124 STDMETHODIMP CKCUser::get_Address2(VARIANT *pVal)
125 {
126     VariantInit(pVal);
127     *pVal = _variant_t(m_vAddress2).Detach();
128     return S_OK;
129 }
130
131 STDMETHODIMP CKCUser::put_Address2(VARIANT newVal)
132 {
```



```
133     m_vAddress2 = newVal;
134     return S_OK;
135 }
136
137 STDMETHODIMP CKCUser::get_City(VARIANT *pVal)
138 {
139     VariantInit(pVal);
140     *pVal = _variant_t(m_vCity).Detach();
141     return S_OK;
142 }
143
144 STDMETHODIMP CKCUser::put_City(VARIANT newVal)
145 {
146     m_vCity = newVal;
147     return S_OK;
148 }
149
150 STDMETHODIMP CKCUser::get_State(VARIANT *pVal)
151 {
152     VariantInit(pVal);
153     *pVal = _variant_t(m_vState).Detach();
154     return S_OK;
155 }
156
157 STDMETHODIMP CKCUser::put_State(VARIANT newVal)
158 {
159     m_vState = newVal;
160     return S_OK;
161 }
162
163 STDMETHODIMP CKCUser::get_Zip(VARIANT *pVal)
164 {
165     VariantInit(pVal);
166     *pVal = _variant_t(m_vZip).Detach();
167     return S_OK;
168 }
169
170 STDMETHODIMP CKCUser::put_Zip(VARIANT newVal)
171 {
172     m_vZip = newVal;
173     return S_OK;
174 }
175
176 STDMETHODIMP CKCUser::get_Phone(VARIANT *pVal)
177 {
178     VariantInit(pVal);
179     *pVal = _variant_t(m_vPhone).Detach();
180     return S_OK;
181 }
182
183 STDMETHODIMP CKCUser::put_Phone(VARIANT newVal)
184 {
185     m_vPhone = newVal;
186     return S_OK;
187 }
188
189 STDMETHODIMP CKCUser::get_Password(VARIANT *pVal)
190 {
191     VariantInit(pVal);
192     *pVal = _variant_t(m_vPassword).Detach();
193     return S_OK;
194 }
195
196 STDMETHODIMP CKCUser::put_Password(VARIANT newVal)
197 {
198     m_vPassword = newVal;
```

```
199     return S_OK;
200 }
201
202 STDMETHODIMP CKCUser::get_Email(VARIANT *pVal)
203 {
204     VariantInit(pVal);
205     *pVal = _variant_t(m_vEmail).Detach();
206     return S_OK;
207 }
208
209 STDMETHODIMP CKCUser::put_Email(VARIANT newVal)
210 {
211     m_vEmail = newVal;
212     return S_OK;
213 }
214
215 STDMETHODIMP CKCUser::put_LCPassword(VARIANT newVal)
216 {
217     m_vLCPassword = newVal;
218     return S_OK;
219 }
220
221 STDMETHODIMP CKCUser::put_LCUser(VARIANT newVal)
222 {
223     m_vLCUser = newVal;
224     return S_OK;
225 }
226
227 STDMETHODIMP CKCUser::get_LCUser(VARIANT *pVal)
228 {
229     VariantInit(pVal);
230     *pVal = _variant_t(m_vLCUser).Detach();
231     return S_OK;
232 }
233
234 STDMETHODIMP CKCUser::get_UserStatus(VARIANT *pVal)
235 {
236     VariantInit(pVal);
237     *pVal = _variant_t(m_vUserStatus).Detach();
238     return S_OK;
239 }
240
241 STDMETHODIMP CKCUser::put_UserStatus(VARIANT newVal)
242 {
243     m_vUserStatus = newVal;
244     return S_OK;
245 }
246
247 STDMETHODIMP CKCUser::update()
248 {
249     Crs_kcuser rs;
250     string strError;
251     HRESULT hr = S_OK;
252
253     rs.setActiveCommand("uptUser");
254     rs.setParameter("kiosk_user_id", m_vKioskUserID);
255     rs.setParameter("first_name", m_vFirstName);
256     rs.setParameter("last_name", m_vLastName);
257     rs.setParameter("middle_name", m_vMiddleName);
258     rs.setParameter("address1", m_vAddress1);
259     rs.setParameter("address2", m_vAddress2);
260     rs.setParameter("city", m_vCity);
261     rs.setParameter("state", m_vState);
262     rs.setParameter("zip", m_vZip);
263     rs.setParameter("email", m_vEmail);
264     rs.setParameter("phone", m_vPhone);
```

```
265     rs.setParameter("password", m_vPassword);
266     rs.setParameter("user_status", m_vUserStatus);
267
268     if (!m_pobjOwner->m_pconn->execute(rs))
269     {
270         m_pobjOwner->m_pconn->getLastError(strError);
271         Error(strError.c_str(), IID_IKUser, hr = E_FAIL);
272     }
273
274     return hr;
275 }
276
277 STDMETHODCALLTYPE CKCUser::addBPReading(VARIANT vSystolicBP, VARIANT vDiastolicBP, VARIANT vPulse)
278 {
279     HRESULT hr = S_OK;
280
281     try
282     {
283         SYSTEMTIME tm;
284         GetLocalTime(&tm);
285         DATE dateNow;
286         SystemTimeToVariantTime(&tm, &dateNow);
287
288         Crs_blood_pressure rs;
289
290         rs.setActiveCommand("insNewReading");
291         rs.setParameter("kiosk_id", _variant_t(m_pobjOwner->m_lKioskID));
292         rs.setParameter("kiosk_user_id", m_vKioskUserID);
293         rs.setParameter("reading_dt", _variant_t(dateNow));
294         rs.setParameter("systolic_bp", _variant_t(vSystolicBP));
295         rs.setParameter("diastolic_bp", _variant_t(vDiastolicBP));
296         rs.setParameter("pulse", _variant_t(vPulse));
297
298         if (!m_pobjOwner->m_pconn->execute(rs))
299         {
300             string strError;
301             m_pobjOwner->m_pconn->getLastError(strError);
302             Error(strError.c_str(), IID_IKData, E_FAIL);
303             throw E_FAIL;
304         }
305
306         removeOldReadings(rs);
307     }
308     catch(_com_error & e)
309     {
310         Error((BSTR) e.Description(), IID_IKUser, hr = e.Error());
311     }
312     catch(HRESULT hrError)
313     {
314         hr = hrError;
315     }
316     catch(...)
317     {
318         hr = E_FAIL;
319     }
320
321     return hr;
322 }
323
324 STDMETHODCALLTYPE CKCUser::addWeightReading(VARIANT vWeight)
325 {
326     HRESULT hr = S_OK;
327
328     try
329     {
```

```
330     SYSTEMTIME tm;
331     GetLocalTime(&tm);
332     DATE dateNow;
333     SystemTimeToVariantTime(&tm, &dateNow);
334
335     Crs_weight      rs;
336
337     rs.setActiveCommand("insNewReading");
338     rs.setParameter("kiosk_id", _variant_t(m_pobjOwner->m_lKioskId));
339     rs.setParameter("kiosk_user_id", m_vKioskUserID);
340     rs.setParameter("reading_dt", _variant_t(dateNow));
341     rs.setParameter("weight", _variant_t(vWeight));
342
343     if (!m_pobjOwner->m_pconn->execute(rs))
344     {
345         string strError;
346         m_pobjOwner->m_pconn->getLastError(strError);
347         Error(strError.c_str(), IID_IKCData, E_FAIL);
348         throw E_FAIL;
349     }
350
351     removeOldReadings(rs);
352 }
353 catch(_com_error & e)
354 {
355     Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
356 }
357 catch(HRESULT hrError)
358 {
359     hr = hrError;
360 }
361 catch(...)
362 {
363     hr = E_FAIL;
364 }
365
366 return hr;
367 }
368
369
370 STDMETHODCALLTYPE CKCUser::addAlternateID(VARIANT vID, VARIANT vIdType)
371 {
372     HRESULT hr = S_OK;
373
374     try
375     {
376         Crs_alternate_id      rs;
377
378         rs.setActiveCommand("insNewId");
379         rs.setParameter("kiosk_id", _variant_t(m_pobjOwner->m_lKioskId));
380         rs.setParameter("alternate_id", _variant_t(vID));
381         rs.setParameter("kiosk_user_id", m_vKioskUserID);
382         if (vIdType.vt == VT_EMPTY || vIdType.vt == VT_NULL)
383             rs.setParameter("id_type", _variant_t(2L));
384         else
385             rs.setParameter("id_type", _variant_t(vIdType));
386
387         if (!m_pobjOwner->m_pconn->execute(rs))
388         {
389             string strError;
390             m_pobjOwner->m_pconn->getLastError(strError);
391             Error(strError.c_str(), IID_IKCData, E_FAIL);
392             throw E_FAIL;
393         }
394     }
395 }
```

```
396     catch(_com_error & e)
397     {
398         Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
399     }
400     catch(HRESULT hrError)
401     {
402         hr = hrError;
403     }
404     catch(...)
405     {
406         hr = E_FAIL;
407     }
408
409     return hr;
410 }
411
412 STDMETHODCALLTYPE CKCUser::getFirstBP(VARIANT *pBPReading)
413 {
414     HRESULT hr = S_OK;
415
416     try
417     {
418         if (m_prsBP != NULL)
419             delete m_prsBP;
420
421         m_prsBP = new Crs_blood_pressure;
422         m_prsBP->setActiveCommand("getReadings");
423         m_prsBP->setParameter("kiosk_user_id", m_vKioskUser.ID);
424
425         if (!m_pobjOwner->m_pconn->execute(*m_prsBP))
426         {
427             string strError;
428             m_pobjOwner->m_pconn->getLastError(strError);
429             Error(strError.c_str(), IID_IKCUser, hr = E_FAIL);
430             throw E_FAIL;
431         }
432
433         hr = getNextBP(pBPReading);
434     }
435     catch(_com_error & e)
436     {
437         Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
438     }
439     catch(HRESULT hrError)
440     {
441         hr = hrError;
442     }
443     catch(...)
444     {
445         hr = E_FAIL;
446     }
447
448     return hr;
449 }
450
451 STDMETHODCALLTYPE CKCUser::getNextBP(VARIANT *pvBPReading)
452 {
453     HRESULT hr = S_OK;
454     VariantClear(pvBPReading);
455     CComObject<CBPReading> * pobjBP = NULL;
456     _variant_t vVal((IDispatch *) NULL, false);
457
458     try
459     {
460         if (m_prsBP == NULL)
461         {
```

```
462         Error("CKCUser::getFirstBP() must be called first", IID_IKCUser, E_FAIL);
463         throw E_FAIL;
464     }
465
466     if (!m_prsBP->isEOF())
467     {
468         if (FAILED(hr = CComObject<CBPReading>::CreateInstance(&pobjBP)))
469         {
470             stringstream strmError;
471             strmError << "CComObject<CBPReading>::CreateInstance() failed. Error =\n
[0x";
472             strmError << std::hex << hr << "]\n";
473             Error(strmError.str().c_str(), IID_IKCUser, hr);
474             throw hr;
475         }
476
477         pobjBP->load(*m_prsBP);
478         m_prsBP->moveNext();
479
480         IDispatch * pIDispatch;
481         if (FAILED(hr = pobjBP->QueryInterface(IID_IDispatch, (void **) &
pIDispatch)))
482         {
483             stringstream strmError;
484             strmError << "QueryInterface(IDispatch) failed. Error = [0x";
485             strmError << std::hex << hr << "]\n";
486             Error(strmError.str().c_str(), IID_IKCUser, hr);
487             pobjBP->Release();
488             throw hr;
489         }
490
491         vVal.pdispVal = pIDispatch;
492     }
493 }
494 catch(_com_error & e)
495 {
496     Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
497 }
498 catch(HRESULT hrError)
499 {
500     hr = hrError;
501 }
502 catch(...)
503 {
504     hr = E_FAIL;
505 }
506
507 if (SUCCEEDED(hr))
508     *pvBPReading = vVal.Detach();
509
510 return hr;
511 }
512
513 STDMETHODCALLTYPE CKCUser::getFirstWeight(VARIANT * pvWeight)
514 {
515     HRESULT hr = S_OK;
516
517     try
518     {
519         if (m_prsWeight != NULL)
520             delete m_prsWeight;
521
522         m_prsWeight = new Crs_weight;
523         m_prsWeight->setActiveCommand("getReadings");
524         m_prsWeight->setParameter("kiosk_user_id", m_vKioskUserID);
525     }
```

```
526         if (!m_pobjOwner->m_pconn->execute(*m_prsWeight))
527         {
528             string strError;
529             m_pobjOwner->m_pconn->getLastError(strError);
530             Error(strError.c_str(), IID_IKCUser, hr = E_FAIL);
531             throw E_FAIL;
532         }
533
534         hr = getNextWeight(pvWeight);
535     }
536     catch(_com_error & e)
537     {
538         Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
539     }
540     catch(HRESULT hrError)
541     {
542         hr = hrError;
543     }
544     catch(...)
545     {
546         hr = E_FAIL;
547     }
548
549     return hr;
550 }
551
552 STDMETHODIMP CKCUser::getNextWeight(VARIANT *pvWeight)
553 {
554     HRESULT hr = S_OK;
555     VariantClear(pvWeight);
556     CComObject<CWeightReading> * pobjWeight = NULL;
557     _variant_t vVal((IDispatch *) NULL, false);
558
559     try
560     {
561         if (m_prsWeight == NULL)
562         {
563             Error("CKCUser::getFirstWeight() must be called first", IID_IKCUser,
564                 E_FAIL);
565             throw E_FAIL;
566         }
567         if (!m_prsWeight->isEOF())
568         {
569             if (FAILED(hr = CComObject<CWeightReading>::CreateInstance(&pobjWeight)))
570             {
571                 stringstream strmError;
572                 strmError << "CComObject<CWeightReading>::CreateInstance() failed.
573                 Error = [0x";
574                 strmError << std::hex << hr << "]";
575                 Error(strmError.str().c_str(), IID_IKCUser, hr);
576                 throw hr;
577             }
578             pobjWeight->load(*m_prsWeight);
579             m_prsWeight->moveNext();
580
581             IDispatch * pIDispatch;
582             if (FAILED(hr = pobjWeight->QueryInterface(IID_IDispatch, (void **)&
583                 pIDispatch)))
584             {
585                 stringstream strmError;
586                 strmError << "QueryInterface(IDispatch) failed. Error = [0x";
587                 strmError << std::hex << hr << "]";
588                 Error(strmError.str().c_str(), IID_IKCUser, hr);
589                 pobjWeight->Release();
590             }
591             vVal = pIDispatch;
592             pIDispatch->Release();
593         }
594     }
595     catch(...)
596     {
597         hr = E_FAIL;
598     }
599
600     *pvWeight = vVal;
601     return hr;
602 }
```

```
589         throw hr;
590     }
591     vVal.pdispVal = pIDispatch;
592 }
593 }
594 }
595 catch(_com_error & e)
596 {
597     Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
598 }
599 catch(HRESULT hrError)
600 {
601     hr = hrError;
602 }
603 catch(...)
604 {
605     hr = E_FAIL;
606 }
607
608 if (SUCCEEDED(hr))
609     *pvWeight = vVal.Detach();
610
611 return hr;
612 }
613
614 STDMETHODIMP CKCUser::validatePassword(VARIANT vPassword, VARIANT vPWordType, VARIANT v
        *pvfValid)
615 {
616     HRESULT hr = S_OK;
617     VariantInit(pvfValid);
618
619     try
620     {
621         pvfValid->vt = VT_BOOL;
622         pvfValid->bVal = 0;
623
624         long lPWordType;
625         if (vPWordType.vt == VT_EMPTY || vPWordType.vt == VT_NULL)
626             lPWordType = 1;
627         else
628             lPWordType = (long) _variant_t(vPWordType);
629
630         if (lPWordType != 1 && lPWordType != 2)
631         {
632             Error("Invalid PasswordType.", IID_IKCUser, E_INVALIDARG);
633             hr = E_INVALIDARG;
634         }
635         else
636         {
637             _bstr_t bstrPWordIn(vPassword);
638             _bstr_t bstrPWord;
639             if (lPWordType == 1)
640                 bstrPWord = m_vLCPassword;
641             else
642                 bstrPWord = m_vPassword;
643
644             string strBPWordIn;
645             string strPWord;
646
647             if (bstrPWordIn.length())
648                 strBPWordIn = (char *) bstrPWordIn;
649
650             if (bstrPWord.length())
651                 strPWord = (char *) bstrPWord;
652
653             TOUPPER(strBPWordIn);
```



```
654         TOUPPER(strPWord);
655
656         if (strBPWordIn.compare(strPWord) == 0)
657             pvfValid->boolVal = -1;
658         else
659             pvfValid->boolVal = 0;
660     }
661 }
662 catch(_com_error & e)
663 {
664     Error((BSTR) e.Description(), IID_IKCUser, hr = e.Error());
665 }
666 catch(...)
667 {
668     Error("Unknown exception", IID_IKCUser, hr = E_FAIL);
669 }
670
671 return hr;
672 }
673
674 //////////////////////////////////////
675 // internal C++ interface
676
677 bool CKCUser::load(CSdoRecordset & rs)
678 {
679     m_vKioskUserID = rs.getField("kiosk_user_id");
680     m_vFirstName = rs.getField("first_name");
681     m_vLastName = rs.getField("last_name");
682     m_vMiddleName = rs.getField("middle_name");
683     m_vAddress1 = rs.getField("address1");
684     m_vAddress2 = rs.getField("address2");
685     m_vCity = rs.getField("city");
686     m_vState = rs.getField("state");
687     m_vZip = rs.getField("zip");
688     m_vPhone = rs.getField("phone");
689     m_vPassword = rs.getField("password");
690     m_vUserStatus = rs.getField("user_status");
691
692     return true;
693 }
694
695 void CKCUser::setOwner(CKCData * pObjOwner)
696 {
697     m_pObjOwner = pObjOwner;
698     m_pObjOwner->AddRef();
699     return;
700 }
701
702 void CKCUser::removeOldReadings(CSdoRecordset & rs)
703 {
704     _variant_t vLastReading;
705     int nNumber = 0;
706     string strError;
707
708     rs.setActiveCommand("getReadings");
709     rs.setParameter("kiosk_user_id", m_vKioskUserID);
710     if (!m_pObjOwner->m_pconn->execute(rs))
711     {
712         m_pObjOwner->m_pconn->getLastError(strError);
713         Error(strError.c_str(), IID_IKCData, E_FAIL);
714         throw E_FAIL;
715     }
716
717     while (!rs.isEOF())
718     {
719         nNumber++;
720     }
721 }
```

```
720         if (nNumber >= m_pobjOwner->m_lReadingsToKeep)
721         {
722             vLastReading = rs.getField("reading_dt");
723             break;
724         }
725         rs.moveNext();
726     }
727
728     rs.close();
729
730     if (vLastReading.vt != VT_EMPTY)
731     {
732         rs.setActiveCommand("removeOld");
733         rs.setParameter("reading_dt", vLastReading);
734         if (!m_pobjOwner->m_pconn->execute(rs))
735         {
736             m_pobjOwner->m_pconn->getLastError(strError);
737             Error(strError.c_str(), IID_IKCData, E_FAIL);
738             throw E_FAIL;
739         }
740     }
741
742     return;
743 }
744
745
```

```

1 // CoBPReading.cpp : Implementation of CBPReading
2 #include "stdafx.h"
3 #include "KCData.h"
4 #include "CoBPReading.h"
5 #include "rs_kcdata.h"
6
7 ///////////////////////////////////////////////////
8 // CBPReading
9
10 STDMETHODIMP CBPReading::InterfaceSupportsErrorInfo(REFIID riid)
11 {
12     static const IID* arr[] =
13     {
14         IID_ICBPReading
15     };
16     for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
17     {
18         if (InlineIsEqualGUID(*arr[i],riid))
19             return S_OK;
20     }
21     return S_FALSE;
22 }
23
24 HRESULT CBPReading::FinalConstruct()
25 {
26     m_vPulse.vt = VT_NULL;
27     m_vDiastolicBP.vt = VT_NULL;
28     m_vSystolicBP.vt = VT_NULL;
29     m_vReadingDt.vt = VT_NULL;
30     m_vKioskUserID.vt = VT_NULL;
31     return S_OK;
32 }
33
34 void CBPReading::FinalRelease()
35 {
36 }
37
38 STDMETHODIMP CBPReading::get_KioskUserID(VARIANT *pVal)
39 {
40     variant_t vVal(*pVal, false);
41     vVal = m_vKioskUserID;
42     *pVal = vVal.Detach();
43     return S_OK;
44 }
45
46 STDMETHODIMP CBPReading::get_ReadingDate(VARIANT *pVal)
47 {
48     variant_t vVal(*pVal, false);
49     vVal = m_vReadingDt;
50     *pVal = vVal.Detach();
51     return S_OK;
52 }
53
54 STDMETHODIMP CBPReading::put_ReadingDate(VARIANT newVal)
55 {
56     m_vReadingDt = newVal;
57     return S_OK;
58 }
59
60 STDMETHODIMP CBPReading::get_SystolicBP(VARIANT *pVal)
61 {
62     variant_t vVal(*pVal, false);
63     vVal = m_vSystolicBP;
64     *pVal = vVal.Detach();
65     return S_OK;
66 }

```

```
67
68 STDMETHODIMP CBPreReading::put_SystolicBP(VARIANT newVal)
69 {
70     m_vSystolicBP = newVal;
71     return S_OK;
72 }
73
74 STDMETHODIMP CBPreReading::get_DiastolicBP(VARIANT *pVal)
75 {
76     _variant_t vVal(*pVal, false);
77     vVal = m_vDiastolicBP;
78     *pVal = vVal.Detach();
79     return S_OK;
80 }
81
82 STDMETHODIMP CBPreReading::put_DiastolicBP(VARIANT newVal)
83 {
84     m_vDiastolicBP = newVal;
85     return S_OK;
86 }
87
88 STDMETHODIMP CBPreReading::get_Pulse(VARIANT *pVal)
89 {
90     _variant_t vVal(*pVal, false);
91     vVal = m_vPulse;
92     *pVal = vVal.Detach();
93     return S_OK;
94 }
95
96 STDMETHODIMP CBPreReading::put_Pulse(VARIANT newVal)
97 {
98     m_vPulse = newVal;
99     return S_OK;
100 }
101
102 //////////////////////////////////////////////////
103 // internal C++ interface
104 bool CBPreReading::load(CSdoRecordset & rs)
105 {
106     m_vPulse = rs.getField("pulse");
107     m_vDiastolicBP = rs.getField("diastolic_bp");
108     m_vSystolicBP = rs.getField("systolic_bp");
109     m_vReadingDt = rs.getField("reading_dt");
110     m_vKioskUserID = rs.getField("kiosk_user_id");
111     return true;
112 }
113
114
```

```

1 // CoKCData.cpp : Implementation of CKCData
2 #include "stdafx.h"
3 #include "KCData.h"
4 #include "CoKCData.h"
5 #include "CoKCUser.h"
6 #include "registryKCData.h"
7 #include "Encryptor.h"
8
9 #include "rs_kcdata.h"
10
11 HRESULT CKCData::FinalConstruct()
12 {
13     m_lKioskId = 0;
14     m_pconn = NULL;
15     return CoCreateFreeThreadedMarshaler(
16         GetControllingUnknown(), &m_pUnkMarshaler.p);
17 }
18
19 void CKCData::FinalRelease()
20 {
21     close();
22     m_pUnkMarshaler.Release();
23 }
24
25
26 //////////////////////////////////////
27 // CKCData
28
29 STDMETHODCALLTYPE CKCData::InterfaceSupportsErrorInfo(REFIID riid)
30 {
31     static const IID* arr[] =
32     {
33         &IID_IKCData
34     };
35
36     for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
37     {
38         if (InlineIsEqualGUID(*arr[i],riid))
39             return S_OK;
40     }
41
42     return S_FALSE;
43 }
44
45 STDMETHODCALLTYPE CKCData::open()
46 {
47     HRESULT hr = S_OK;
48
49     try
50     {
51         CRegistryKCData registry;
52         if (!registry.open())
53         {
54             Error(registry.m_strLastError.c_str(), IID_IKCData, E_FAIL);
55             throw E_FAIL;
56         }
57
58         m_lKioskId = registry.m_lKioskId;
59         m_lReadingsToKeep = registry.m_lNumberReadingsToKeep;
60
61         if (m_pconn != NULL)
62             delete m_pconn;
63
64         m_pconn = new CSdoConnection;
65
66         // these options are not compatible with access driver

```

```
67     m_pconn->setCommitOptionsOnConnect(false);
68     m_pconn->useProvider(false);
69     m_pconn->setForwardRecordsetToOpen(false);
70
71     string strConn = "DSN=";
72     strConn += registry.m_strKCDataSource + ";";
73
74     if (!m_pconn->connect( strConn.c_str(),
75                          registry.m_strKCUser.c_str(),
76                          registry.m_strKCPasswd.c_str() ))
77     {
78         string strError;
79         m_pconn->getLastError(strError);
80         Error(strError.c_str(), IID_IKCData, E_FAIL);
81         throw E_FAIL;
82     }
83 }
84 catch(HRESULT hrError)
85 {
86     hr = hrError;
87 }
88 catch(...)
89 {
90     hr = E_FAIL;
91 }
92
93 return hr;
94 }
95
96 STDMETHODCALLTYPE CKCData::close()
97 {
98     if (m_pconn)
99     {
100         m_pconn->close();
101         delete m_pconn;
102     }
103
104     m_pconn = NULL;
105     return S_OK;
106 }
107
108 STDMETHODCALLTYPE CKCData::buildRegistry()
109 {
110     HRESULT hr = S_OK;
111
112     CRegistryKCData registry;
113     if (!registry.buildInitial())
114         Error(registry.m_strLastError.c_str(), IID_IKCData, hr = E_FAIL);
115
116     return hr;
117 }
118
119 STDMETHODCALLTYPE CKCData::getKioskUser(VARIANT vKioskId, VARIANT vIdType, VARIANT *
120     pvDispUser)
121 {
122     VariantInit(pvDispUser);
123     _variant_t vRetVal({IDispatch *} NULL, false);
124
125     HRESULT hr = S_OK;
126
127     try
128     {
129         CIs_kcdata rs;
130         rs.setActiveCommand("cmdGetById");
131         rs.setParameter("alternate_id", _variant_t(vKioskId));
132         rs.setParameter("id_type", _variant_t(vIdType));
```

```
132     if (!m_pconn->execute(rs))
133     {
134         string strError;
135         m_pconn->getLastError(strError);
136         Error(strError.c_str(), IID_IKCData, hr = E_FAIL);
137         throw hr;
138     }
139
140     if (!rs.isEmpty())
141     {
142         CComObject<CKCUser> * pobjUser;
143
144         if (FAILED(hr = CComObject<CKCUser>::CreateInstance(&pobjUser)))
145         {
146             Error("CComObject<CKCUser>::CreateInstance() failed.", IID_IKCUser, hr);
147             throw hr;
148         }
149
150         pobjUser->setOwner(this);
151         IDispatch * pIDisp = NULL;
152
153         if (FAILED(hr = pobjUser->QueryInterface(IID_IDispatch, (void **)&pIDisp)))
154         {
155             Error("QueryInterface(IDispatch *) failed", IID_IKCUser, hr);
156             pobjUser->Release();
157             throw hr;
158         }
159
160         vRetVal = pIDisp;
161
162         pobjUser->load(rs);
163     }
164 }
165 catch(_com_error & e)
166 {
167     _bstr_t bstrError = e.Description();
168     Error((char *) bstrError, IID_IKCData, hr = e.Error());
169 }
170 catch(HRESULT hrError)
171 {
172     hr = hrError;
173 }
174 catch(...)
175 {
176     hr = E_FAIL;
177 }
178
179 *pvDispUser = vRetVal;
180
181 return hr;
182 }
183
184 STDMETHODIMP CKCData::getLCUser(VARIANT vLogin, VARIANT *pvUser)
185 {
186     VariantInit(pvUser);
187
188     _variant_t vRetVal((IDispatch *) NULL, false);
189
190     HRESULT hr = S_OK;
191
192     try
193     {
194         // check to see if LC user
195         Crs_lifeclinic_users rsLC;
```

```

196     rsLC.setActiveCommand("getByName");
197     rsLC.setParameter("user_name", _variant_t(vLogon));
198     if (!m_pconn->execute(rsLC))
199     {
200         string strError;
201         m_pconn->getLastError(strError);
202         Error(strError.c_str(), IID_IKCData, hr = E_FAIL);
203         throw hr;
204     }
205
206     _variant_t vUserName;
207     _variant_t vUserPassword;
208     _variant_t vCpiId;
209     _variant_t vUser;
210
211     if (!rsLC.isEmpty())
212     {
213         vCpiId = rsLC.getField("lifeclinic_id");
214         string strEncryptedPassword;
215         string strPassword;
216         rsLC.getField("password", strEncryptedPassword);
217         if (strEncryptedPassword.size())
218         {
219             CEncryptor encryptor;
220             encryptor.Decrypt(strEncryptedPassword.c_str(), NULL, strPassword);
221             vUserPassword = strPassword.c_str();
222         }
223         vUserName = rsLC.getField("user_name");
224         rsLC.close();
225
226         if (FAILED(getKioskUser(vCpiId, _variant_t((long) IdType_LC), &vRetVal)))
227             throw E_FAIL;
228
229         IKCUser * pIKCUser = NULL;
230         bool fNewUser = false;
231         if (vRetVal.pdispVal == NULL)
232         {
233             if (FAILED(createUser(&vRetVal)))
234                 throw E_FAIL;
235
236             fNewUser = true;
237         }
238
239         if (FAILED(hr = vRetVal.pdispVal->QueryInterface(IID_IKCUser, (void **)&
pIKCUser)))
240         {
241             stringstream strmError;
242             strmError << "pIDispatch->QueryInterface(IID_IKCUser) failed. Error = ";
243             strmError << std::hex << hr << "].";
244             Error(strmError.str().c_str(), IID_IKCData, hr);
245             throw hr;
246         }
247
248         if (fNewUser)
249         {
250             pIKCUser->addAlternateID(vCpiId, _variant_t((long) IdType_LC));
251             pIKCUser->put_userstatus(_variant_t((long) USTAT_NEWKCU));
252             pIKCUser->update();
253         }
254
255         pIKCUser->put_LCUser(vUserName);
256         pIKCUser->put_LCPassword(vUserPassword);
257         pIKCUser->Release();
258     }
259

```



```
260     catch(_com_error & e)
261     {
262         bstr_t bstrError = e.Description();
263         Error((char *) bstrError, IID_IKCData, hr = e.Error());
264     }
265     catch(HRESULT hrError)
266     {
267         hr = hrError;
268     }
269     catch(...)
270     {
271         hr = E_FAIL;
272     }
273
274     *pvUser = vRetVal.Detach();
275
276     return hr;
277 }
278
279 STDMETHODCALLTYPE CKCData::createUser(VARIANT * pvUser)
280 {
281     HRESULT hr = S_OK;
282     bool fTransBegun = false;
283     string strError;
284     CComObject<CKCUser> * pObjUser = NULL;
285
286     try
287     {
288         if (m_pconn == NULL || !m_pconn->isConnected())
289         {
290             Error("There is no database connection", IID_IKCData, hr = E_FAIL);
291             throw hr;
292         }
293
294         //////////////////////////////////////
295         // create underlying database record
296         fTransBegun = m_pconn->beginTransaction();
297
298         Crs_kcuser rs;
299         rs.setActiveCommand("cmdInsNewUser");
300         rs.setParameter("kiosk_id", _variant_t(m_lKioskId));
301         if (!m_pconn->execute(rs))
302         {
303             m_pconn->getLastError(strError);
304             Error(strError.c_str(), IID_IKCData, E_FAIL);
305             throw E_FAIL;
306         }
307
308         // get the newly created id
309         rs.setActiveCommand("cmdGetNewUser");
310         if (!m_pconn->execute(rs))
311         {
312             m_pconn->getLastError(strError);
313             Error(strError.c_str(), IID_IKCData, E_FAIL);
314             throw E_FAIL;
315         }
316         _variant_t vNewId = rs.getField("kiosk_user_id");
317         rs.close();
318
319         // mark record as in use
320         rs.setActiveCommand("setUserInUse");
321         rs.setParameter("in_use", _variant_t(-1L));
322         rs.setParameter("kiosk_user_id", vNewId);
323         if (!m_pconn->execute(rs))
324         {
325
```

```

326         m_pconn->getLastError(strError);
327         Error(strError.c_str(), IID_IKData, E_FAIL);
328         throw E_FAIL;
329     }
330
331     // put primary id in id map
332     rs.setActiveCommand("putIdMap");
333     rs.setParameter("kiosk_id", _variant_t(m_lKioskId));
334     rs.setParameter("alternate_id", vNewId);
335     rs.setParameter("kiosk_user_id", vNewId);
336     if (m_pconn->execute(rs);
337     {
338         m_pconn->getLastError(strError);
339         Error(strError.c_str(), IID_IKData, E_FAIL);
340         throw E_FAIL;
341     }
342
343     //////////////////////////////////////////
344     // create com object
345     if (FAILED(hr = CComObject<CKCUser>::CreateInstance(&pobjUser)))
346     {
347         Error("CComObject<CKCUser>::CreateInstance() failed", IID_IKData, hr);
348         throw hr;
349     }
350     pobjUser->setOwner(this);
351     pobjUser->m_vKioskUserID = vNewId;
352
353
354     // return IDispatch in VARIANT
355     IDispatch * pIDisp = NULL;
356     if (FAILED(hr = pobjUser->QueryInterface(IID_IDispatch, (void**) &pIDisp)))
357     {
358         Error("CComObject<CKCUser>::QueryInterface(IDispatch) failed", IID_IKData,
359         , hr);
360         throw hr;
361     }
362     _variant_t vRet(pIDisp, false);
363     *pvUser = vRet.Detach();
364
365     pobjUser = NULL;
366 }
367 catch(_com_error & e)
368 {
369     _bstr_t bstrError = e.Description();
370     Error((char *) bstrError, IID_IKData, hr = e.Error());
371 }
372 catch(HRESULT hrError)
373 {
374     hr = hrError;
375 }
376 catch(...)
377 {
378     Error("Unknown exception", IID_IKData, hr = E_FAIL);
379 }
380
381 if (fTransBegun)
382 {
383     if (SUCCEEDED(hr))
384         m_pconn->commitTrans();
385     else
386         m_pconn->rollbackTrans();
387 }
388
389 // if pobjUser is valued then an error occurred after its creation
390 if (pobjUser != NULL)

```

```
391     delete pObjUser;
392
393     return hr;
394 }
395
396 STDMETHODCALLTYPE CKCData::getUnexportedData(VARIANT *pvData)
397 {
398     HRESULT hr = S_OK;
399
400     VariantInit(pvData);
401     string strError;
402
403     try
404     {
405         Crs_kcuser rsUser;
406         Crs_blood_pressure rsBP;
407         Crs_weight rsWeight;
408         Crs_alternate_id rsIds;
409
410         rsUser.setActiveCommand("getUnExported");
411         rsBP.setActiveCommand("getUnExported");
412         rsWeight.setActiveCommand("getUnExported");
413         rsIds.setActiveCommand("getUnExported");
414
415         if (!m_pconn->execute(rsUser))
416         {
417             m_pconn->getLastError(strError);
418             Error(strError.c_str(), IID_IKCData, E_FAIL);
419             throw E_FAIL;
420         }
421
422         if (!m_pconn->execute(rsBP))
423         {
424             m_pconn->getLastError(strError);
425             Error(strError.c_str(), IID_IKCData, E_FAIL);
426             throw E_FAIL;
427         }
428
429         if (!m_pconn->execute(rsWeight))
430         {
431             m_pconn->getLastError(strError);
432             Error(strError.c_str(), IID_IKCData, E_FAIL);
433             throw E_FAIL;
434         }
435
436         if (!m_pconn->execute(rsIds))
437         {
438             m_pconn->getLastError(strError);
439             Error(strError.c_str(), IID_IKCData, E_FAIL);
440             throw E_FAIL;
441         }
442
443         CXmlDocument xdocXData("<kiosk_data/>");
444         xdocXData.setUppercaseTags(false);
445
446         rsUser.toXml(xdocXData);
447         rsBP.toXml(xdocXData);
448         rsWeight.toXml(xdocXData);
449         rsIds.toXml(xdocXData);
450
451         string strXml;
452         xdocXData.getXML(strXml);
453
454         _variant_t vVal = strXml.c_str();
455         *pvData = vVal.Detach();
456     }
```

```
457     catch(_com_error & e)
458     {
459         Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
460     }
461     catch(HRESULT hrError)
462     {
463         hr = hrError;
464     }
465     catch(...)
466     {
467         hr = E_FAIL;
468     }
469     return hr;
470 }
471
472
473 STDMETHODIMP CKCData::markDataExported()
474 {
475     HRESULT hr = S_OK;
476     string strError;
477     bool fTransStarted = false;
478
479     try
480     {
481         Crs_kruser rsUser;
482         Crs_blood_pressure rsBP;
483         Crs_weight rsWeight;
484         Crs_alterate_id rsIds;
485
486         rsUser.setActiveCommand("markExported");
487         rsBP.setActiveCommand("markExported");
488         rsWeight.setActiveCommand("markExported");
489         rsIds.setActiveCommand("markExported");
490
491         if (!fTransStarted = m_pconn->beginTransaction())
492         {
493             m_pconn->getLastError(strError);
494             Error(strError.c_str(), IID_IKCData, E_FAIL);
495             throw E_FAIL;
496         }
497
498         if (!m_pconn->execute(rsUser))
499         {
500             m_pconn->getLastError(strError);
501             Error(strError.c_str(), IID_IKCData, E_FAIL);
502             throw E_FAIL;
503         }
504
505         if (!m_pconn->execute(rsBP))
506         {
507             m_pconn->getLastError(strError);
508             Error(strError.c_str(), IID_IKCData, E_FAIL);
509             throw E_FAIL;
510         }
511
512         if (!m_pconn->execute(rsWeight))
513         {
514             m_pconn->getLastError(strError);
515             Error(strError.c_str(), IID_IKCData, E_FAIL);
516             throw E_FAIL;
517         }
518
519         if (!m_pconn->execute(rsIds))
520         {
521             m_pconn->getLastError(strError);
522             Error(strError.c_str(), IID_IKCData, E_FAIL);
```

```
523         throw E_FAIL;
524     }
525 }
526 catch(_com_error & e)
527 {
528     Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
529 }
530 catch(HRESULT hrError)
531 {
532     hr = hrError;
533 }
534 catch(...)
535 {
536     hr = E_FAIL;
537 }
538
539 if (fTransStarted)
540 {
541     if (SUCCEEDED(hr))
542         m_pconn->commitTrans();
543     else
544         m_pconn->rollbackTrans();
545 }
546
547 return hr;
548 }
549
550 STDMETHODCALLTYPE CKCData::markDataUnexported()
551 {
552     HRESULT hr = S_OK;
553     string strError;
554     bool fTransStarted = false;
555
556     try
557     {
558         Crs_kcuser rsUser;
559         Crs_blood_pressure rsBP;
560         Crs_weight rsWeight;
561         Crs_alternate_id rsIds;
562
563         rsUser.setActiveCommand("markUnexported");
564         rsBP.setActiveCommand("markUnexported");
565         rsWeight.setActiveCommand("markUnexported");
566         rsIds.setActiveCommand("markUnexported");
567
568         if (!(fTransStarted = m_pconn->beginTrans()))
569         {
570             m_pconn->getLastError(strError);
571             Error(strError.c_str(), IID_IKCData, E_FAIL);
572             throw E_FAIL;
573         }
574
575         if (!m_pconn->execute(rsUser))
576         {
577             m_pconn->getLastError(strError);
578             Error(strError.c_str(), IID_IKCData, E_FAIL);
579             throw E_FAIL;
580         }
581
582         if (!m_pconn->execute(rsBP))
583         {
584             m_pconn->getLastError(strError);
585             Error(strError.c_str(), IID_IKCData, E_FAIL);
586             throw E_FAIL;
587         }
588     }
```

```

589         if (!m_pconn->execute(rsWeight));
590     {
591         m_pconn->getLastError(strError);
592         Error(strError.c_str(), IID_IKCData, E_FAIL);
593         throw E_FAIL;
594     }
595
596     if (!m_pconn->execute(rsIds))
597     {
598         m_pconn->getLastError(strError);
599         Error(strError.c_str(), IID_IKCData, E_FAIL);
600         throw E_FAIL;
601     }
602 }
603 catch(_com_error & e)
604 {
605     Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
606 }
607 catch(HRESULT hrError)
608 {
609     hr = hrError;
610 }
611 catch(...)
612 {
613     hr = E_FAIL;
614 }
615
616 if (fTransStarted)
617 {
618     if (SUCCEEDED(hr))
619         m_pconn->commitTrans();
620     else
621         m_pconn->rollbackTrans();
622 }
623
624 return hr;
625 }
626
627 STDMETHODIMP CKCData::importLCUsers(VARIANT vFileName)
628 {
629     HRESULT hr = S_OK;
630
631     try
632     {
633         string strFileName = (char *) (_bstr_t) vFileName;
634
635         CXmIDocument xdocLCUsers;
636         if (!loadXmlFile(strFileName.c_str(), xdocLCUsers, false))
637             throw E_FAIL;
638
639         CXmlElement &elTable;
640
641         if (xdocLCUsers.getItem("cpi_user", &elTable))
642         {
643             xdocLCUsers.pushCurrent(&elTable);
644             Crs_lifeclinic users rsLCUsers;
645             if (!applyTable(xdocLCUsers, rsLCUsers, FLAG_INSERT | FLAG_UPDATE))
646                 throw E_FAIL;
647             xdocLCUsers.popCurrent();
648         }
649     }
650     catch(_com_error & e)
651     {
652         Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
653     }
654     catch(HRESULT hrError)

```

```
655 {
656     hr = hrError;
657 }
658 catch(...)
659 {
660     hr = E_FAIL;
661 }
662
663 return hr;
664 }
665
666
667 STDMETHODIMP CKCData::importData(VARIANT vFileName)
668 {
669     HRESULT hr = S_OK;
670     try
671     {
672         string strFileName = (char *) (_bstr_t) vFileName;
673
674         CXmlDocument docTrans;
675         if (!loadXmlFile(strFileName.c_str(), docTrans, true))
676             throw E_FAIL;
677
678         CXMLElement elTable;
679
680         // apply id map to the database
681         if (docTrans.getItem("kc_id_map", elTable))
682         {
683             docTrans.pushCurrent(elTable);
684             Crc_alternate_id rsIdMap;
685             if (!applyTable(docTrans, rsIdMap, FLAG_INSERT))
686                 throw E_FAIL;
687             docTrans.popCurrent();
688         }
689     }
690     catch(_com_error & e)
691     {
692         Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
693     }
694     catch(HRESULT hrError)
695     {
696         hr = hrError;
697     }
698     catch(...)
699     {
700         hr = E_FAIL;
701     }
702
703     return hr;
704 }
705
706 STDMETHODIMP CKCData::checkKioskUserID(VARIANT vKioskID, VARIANT *pvIDExists)
707 {
708     HRESULT hr = S_OK;
709     bool fSuccess = false;
710
711     try
712     {
713         Crc_kcdata rs;
714         rs.SetActiveCommand("cmdCheckIDExists");
715         rs.SetParameter("alternate_id", _variant_t(vKioskID));
716
717         if (!m_pConn->execute(rs))
718         {
719             string strError;
720
```

```
721         m_pconn->getLastError(strError);
722         Error(strError.c_str(), IID_IKCData, hr = E_FAIL);
723         throw hr;
724     }
725
726     if (!rs.isEmpty())
727         fSuccess = true;
728     else
729         fSuccess = false;
730
731 }
732 catch(_com_error & e)
733 {
734     _bstr_t bstrError = e.Description();
735     Error((char *) bstrError, IID_IKCData, hr = e.Error());
736 }
737 catch(HRESULT hrError)
738 {
739     hr = hrError;
740 }
741 catch(...)
742 {
743     hr = E_FAIL;
744 }
745
746 _variant_t vRetVal = fSuccess;
747 *pvtExists = vRetVal;
748 return hr;
749 }
750
751
752 bool CKCData::applyTable(CXmldocument & xdoc, CSdoRecordset & rs, unsigned short
753 fupdateFlags)
754 {
755     CXmlelement elTable;
756     CXmlelement elRow;
757     xdoc.getCurrent(&elTable);
758     bool fRowFound = elTable.getFirst(&elRow);
759     while(fRowFound)
760     {
761         // check if record exists
762         xdoc.pushCurrent(&elRow);
763         rs.setActiveCommand("applyExists");
764         rs.getActiveCommand()->clearParms();
765         rs.getActiveCommand()->setParmsOnly(xdoc);
766         xdoc.popCurrent();
767         if (!m_pconn->execute(rs))
768         {
769             string strError;
770             m_pconn->getLastError(strError);
771             Error(strError.c_str(), IID_IKCData, E_FAIL);
772             return false;
773         }
774         bool fExists = !rs.isEmpty();
775         rs.close();
776
777         // if row doesn't exist and finserts true then add it
778         if (!fExists && (fUpdateFlags & FLAG_INSERT))
779         {
780             xdoc.pushCurrent(&elRow);
781             rs.setActiveCommand("applyInsert");
782             rs.getActiveCommand()->clearParms();
783             rs.getActiveCommand()->setParmsOnly(xdoc);
784             xdoc.popCurrent();
785             if (!m_pconn->execute(rs))
786             {
```



```
786         string strError;
787         m_pconn->getLastError(strError);
788         Error(strError.c_str(), IID_IKCData, E_FAIL);
789         return false;
790     }
791 }
792
793 // if row exists and fUpdates true then update it
794 if (!exists && (fUpdates & !CLASS_UPDATE))
795 {
796     xdoc.pushCurrent(&elRow);
797     rs.setActiveCommand("applyUpdate");
798     rs.getActiveCommand()->clearParms();
799     rs.getActiveCommand()->setParmsOnly(xdoc);
800     xdoc.popCurrent();
801     if (!m_pconn->execute(rs))
802     {
803         string strError;
804         m_pconn->getLastError(strError);
805         Error(strError.c_str(), IID_IKCData, E_FAIL);
806         return false;
807     }
808 }
809
810 fRowFound = elTable.getNext(&elRow);
811 }
812
813 return true;
814 }
815
816 bool CKCData::loadXmlFile(LPCSTR pszFileName, CXmlDocument & xdocResult, bool
fEncrypted)
817 {
818     bool fSuccess = true;
819
820     stringstream strmError;
821     HRESULT hr = S_OK;
822
823     try
824     {
825         strmError << "CKCData::loadXmlFile():";
826
827         // open import file name
828         FILE * pstream = fopen(pszFileName, "rb");
829         if (pstream == NULL)
830         {
831             strmError << "Unable to open file [" << pszFileName << "].";
832             Error(strmError.str().c_str(), IID_IKCData, E_FAIL);
833             throw E_FAIL;
834         }
835
836         // get the size of the file
837         fseek(pstream, 0, SEEK_END);
838         long lFileSize = ftell(pstream);
839         fseek(pstream, 0, SEEK_SET);
840
841         // allocate file buffer
842         unsigned char * pBuffer = new unsigned char [lFileSize + 1];
843         if (pBuffer == NULL)
844         {
845             strmError << "Unable to allocate file buffer. Out of memory.";
846             Error(strmError.str().c_str(), IID_IKCData, E_FAIL);
847             throw E_FAIL;
848         }
849         pBuffer[lFileSize] = 0;
850     }
```

```
851 // read the import file
852 long lBytesRead = fread(pBuff, 1, lFileSize, pstream);
853 if (lBytesRead != lFileSize)
854 {
855     strmError << "CKCData::importData() failed. Unable to read file [";
856     strmError << pszFileName << "].";
857     Error(strmError.str().c_str(), IID_IKCData, E_FAIL);
858     throw E_FAIL;
859 }
860 // decrypt the file to xml
861 string strXmlData;
862 if (fEncrypted)
863 {
864     CEncryptor encrypt;
865     encrypt.Decrypt((LPCSTR) pBuff, NULL, strXmlData);
866 }
867 else
868     strXmlData = (const char *) pBuff;
869 delete [] pBuff;
870 // parse the xml
871 xdocResult.loadDocument(strXmlData.c_str());
872 if (!xdocResult.isReady())
873 {
874     string strParseError;
875     xdocResult.getParserError(strParseError);
876     strmError << "Unable to parse import file [" << pszFileName << "]. Error =\n";
877     strmError << strParseError << "].";
878     Error(strmError.str().c_str(), IID_IKCData, E_FAIL);
879     throw E_FAIL;
880 }
881 catch(_com_error & e)
882 {
883     Error((BSTR) e.Description(), IID_IKCData, hr = e.Error());
884     fSuccess = false;
885 }
886 catch(HRESULT hrError)
887 {
888     hr = hrError;
889     fSuccess = false;
890 }
891 catch(...)
892 {
893     hr = E_FAIL;
894     fSuccess = false;
895 }
896 return fSuccess;
897 }
```

```
1 // CoWeightReading.cpp : Implementation of CWeightReading
2 #include "stdafx.h"
3 #include "KCDATA.h"
4 #include "CoWeightReading.h"
5 #include "rs_kcdata.h"
6
7 //////////////////////////////////////
8 // CWeightReading
9
10 STDMETHODIMP CWeightReading::InterfaceSupportsErrorInfo(REFIID riid)
11 {
12     static const IID* arr[] =
13     {
14         IID_IWeightReading
15     };
16     for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++)
17     {
18         if (InlineIsEqualGUID(*arr[i],riid))
19             return S_OK;
20     }
21     return S_FALSE;
22 }
23
24 HRESULT CWeightReading::FinalConstruct()
25 {
26     m_vKioskUserID.vt = VT_NULL;
27     m_vWeight.vt = VT_NULL;
28     m_vReadingDate.vt = VT_NULL;
29     return S_OK;
30 }
31
32 void CWeightReading::FinalRelease()
33 {
34     return;
35 }
36
37
38
39 STDMETHODIMP CWeightReading::get_Weight(VARIANT *pVal)
40 {
41     _variant_t vVal(*pVal, false);
42     vVal = m_vWeight;
43     *pVal = vVal.Detach();
44     return S_OK;
45 }
46
47 STDMETHODIMP CWeightReading::put_Weight(VARIANT newVal)
48 {
49     m_vWeight = newVal;
50     return S_OK;
51 }
52
53 STDMETHODIMP CWeightReading::get_KioskUserID(VARIANT *pVal)
54 {
55     _variant_t vVal(*pVal, false);
56     vVal = m_vKioskUserID;
57     *pVal = vVal.Detach();
58     return S_OK;
59 }
60
61 STDMETHODIMP CWeightReading::get_ReadingDate(VARIANT *pVal)
62 {
63     _variant_t vVal(*pVal, false);
64     vVal = m_vReadingDate;
65     *pVal = vVal.Detach();
66     return S_OK;
67 }
```

```
67 }
68
69 STDMETHODIMP CWeightReading::put_ReadingDate(VARIANT newVal)
70 {
71     m_vReadingDate = newVal;
72     return S_OK;
73 }
74
75 //////////////////////////////////////
76 // internal C++ interface
77
78 bool CWeightReading::load(CSdoRecordset & rs)
79 {
80     m_vWeight = rs.getField("weight");
81     m_vReadingDate = rs.getField("reading_dt");
82     m_vKioskUserID = rs.getField("kiosk_user_id");
83     return true;
84 }
85
86
```

```
1 // LCKioskServer.cpp : Implementation of WinMain
2
3
4 // Note: Proxy/Stub Information
5 // To build a separate proxy/stub DLL,
6 // run nmake -f LCKioskServerps.mk in the project directory.
7
8 #include "stdafx.h"
9 #include "resource.h"
10 #include <initguid.h>
11 #include "threadMain.h"
12 #include "LCKioskServer.h"
13 #include "registryKServer.h"
14
15 #include "LCKioskServer_i.c"
16
17
18 #include <stdio.h>
19
20 //////////////////////////////////////
21 // MFC support
22 CKioskServerApp _theApp;
23
24 //////////////////////////////////////
25 // ATL support
26 CServiceModule _Module;
27
28 //////////////////////////////////////
29 //Global decalarations
30 CLogNTEvents _logEvents("Kiosk Server");
31 CLogFile _logFile("c:\\LCKioskServer.log");
32 CLogDebug _logDebug;
33 CLogMulti _logAll;
34
35 BEGIN_OBJECT_MAP(ObjectMap)
36 END_OBJECT_MAP()
37
38
39 LPCTSTR FindOneOf(LPCTSTR p1, LPCTSTR p2)
40 {
41     while (p1 != NULL && *p1 != NULL)
42     {
43         LPCTSTR p = p2;
44         while (p != NULL && *p != NULL)
45         {
46             if (*p1 == *p)
47                 return CharNext(p1);
48             p = CharNext(p);
49         }
50         p1 = CharNext(p1);
51     }
52     return NULL;
53 }
54
55 // Although some of these functions are big they are declared inline since they are
56 // only used once
57
58 inline HRESULT CServiceModule::RegisterServer(BOOL bRegTypeLib, BOOL bService)
59 {
60     HRESULT hr = CoInitialize(NULL);
61     if (FAILED(hr))
62         return hr;
63
64     // Remove any previous service since it may point to
65     // the incorrect file
66     Uninstall();
```

```
66
67 // Add service entries
68 UpdateRegistryFromResource(IDR_LCKioskServer, TRUE);
69
70 // Adjust the AppID for Local Server or Service
71 CRegKey keyAppID;
72 LONG lRes = keyAppID.Open(HKEY_CLASSES_ROOT, _T("AppID"), KEY_WRITE);
73 if (lRes != ERROR_SUCCESS)
74     return lRes;
75
76 CRegKey key;
77 lRes = key.Open(keyAppID, _T("{BF823564-E93B-11D3-B88C-CC792E000000}"), KEY_WRITE);
78 if (lRes != ERROR_SUCCESS)
79     return lRes;
80 key.DeleteValue(_T("LocalService"));
81
82 if (bService)
83 {
84     key.SetValue(_T("LCKioskServer"), _T("LocalService"));
85     key.SetValue(_T("-Service"), _T("ServiceParameters"));
86     // Create service
87     Install();
88 }
89
90 // Add object entries
91 hr = CComModule::RegisterServer(bRegTypeLib);
92
93 CoUninitialize();
94 return hr;
95 }
96
97 inline HRESULT CServiceModule::UnregisterServer()
98 {
99     HRESULT hr = CoInitialize(NULL);
100     if (FAILED(hr))
101         return hr;
102
103     // Remove service entries
104     UpdateRegistryFromResource(IDR_LCKioskServer, FALSE);
105     // Remove service
106     Uninstall();
107     // Remove object entries
108     CComModule::UnregisterServer(TRUE);
109     CoUninitialize();
110     return S_OK;
111 }
112
113 inline void CServiceModule::Init(ATL_OBJMAP_ENTRY* p, HINSTANCE h, UINT
    nServiceNameID, const GUID* plibid)
114 {
115     CComModule::Init(p, h, plibid);
116
117     m_bService = TRUE;
118
119     LoadString(h, nServiceNameID, m_szServiceName, sizeof(m_szServiceName) / sizeof
        (TCHAR));
120
121     // set up the initial service status
122     m_hServiceStatus = NULL;
123     m_status.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
124     m_status.dwCurrentState = SERVICE_STOPPED;
125     m_status.dwControlsAccepted = SERVICE_ACCEPT_STOP;
126     m_status.dwWin32ExitCode = 0;
127     m_status.dwServiceSpecificExitCode = 0;
128     m_status.dwCheckpoint = 0;
```

```
129     m_status.dwWaitHint = 0;
130 }
131
132 LONG CServiceModule::Unlock()
133 {
134     LONG l = CComModule::Unlock();
135     if (l == 0 && !m_bService)
136         PostThreadMessage(dwThreadId, WM_QUIT, 0, 0);
137     return l;
138 }
139
140 BOOL CServiceModule::IsInstalled()
141 {
142     BOOL bResult = FALSE;
143
144     SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
145
146     if (hSCM != NULL)
147     {
148         SC_HANDLE hService = ::OpenService(hSCM, m_szServiceName,
149 SERVICE_QUERY_CONFIG);
150         if (hService != NULL)
151         {
152             bResult = TRUE;
153             ::CloseServiceHandle(hService);
154         }
155         ::CloseServiceHandle(hSCM);
156     }
157     return bResult;
158 }
159
160 inline BOOL CServiceModule::Install()
161 {
162     if (IsInstalled())
163         return TRUE;
164
165     SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
166     if (hSCM == NULL)
167     {
168         MessageBox(NULL, _T("Couldn't open service manager"), m_szServiceName, MB_OK);
169         return FALSE;
170     }
171
172     // Get the executable file path
173     TCHAR szFilePath[_MAX_PATH];
174     ::GetModuleFileName(NULL, szFilePath, MAX_PATH);
175
176     SC_HANDLE hService = ::CreateService(
177 hSCM, m_szServiceName, m_szServiceName,
178 SERVICE_ALL_ACCESS, SERVICE_WIN32_OWN_PROCESS,
179 SERVICE_DEMAND_START, SERVICE_ERROR_NORMAL,
180 szFilePath, NULL, NULL, _T("RPCSS\0"), NULL, NULL);
181
182     if (hService == NULL)
183     {
184         ::CloseServiceHandle(hSCM);
185         MessageBox(NULL, _T("Couldn't create service"), m_szServiceName, MB_OK);
186         return FALSE;
187     }
188
189     ::CloseServiceHandle(hService);
190     ::CloseServiceHandle(hSCM);
191     return TRUE;
192 }
193
194 inline BOOL CServiceModule::Uninstall()
```

```
194 {
195     if (!IsInstalled())
196         return TRUE;
197
198     SC_HANDLE hSCM = ::OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
199
200     if (hSCM == NULL)
201     {
202         MessageBox(NULL, _T("Couldn't open service manager"), m_szServiceName, MB_OK);
203         return FALSE;
204     }
205
206     SC_HANDLE hService = ::OpenService(hSCM, m_szServiceName, SERVICE_STOP | DELETE);
207
208     if (hService == NULL)
209     {
210         ::CloseServiceHandle(hSCM);
211         MessageBox(NULL, _T("Couldn't open service"), m_szServiceName, MB_OK);
212         return FALSE;
213     }
214     SERVICE_STATUS status;
215     ::ControlService(hService, SERVICE_CONTROL_STOP, &status);
216
217     BOOL bDelete = ::DeleteService(hService);
218     ::CloseServiceHandle(hService);
219     ::CloseServiceHandle(hSCM);
220
221     if (bDelete)
222         return TRUE;
223
224     MessageBox(NULL, _T("Service could not be deleted"), m_szServiceName, MB_OK);
225     return FALSE;
226 }
227
228 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
229 // Logging functions
230 void CServiceModule::LogEvent(LPCTSTR pFormat, ...)
231 {
232     TCHAR chMsg[2048];
233     va_list pArg;
234
235     va_start(pArg, pFormat);
236     _vstprintf(chMsg, pFormat, pArg);
237     va_end(pArg);
238
239     CLogMsgEvent(LCEV_GENERIC, -1, chMsg).Post(_logAll);
240 }
241
242 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
243 // Service startup and registration
244 inline void CServiceModule::Start()
245 {
246     SERVICE_TABLE_ENTRY st[] =
247     {
248         { m_szServiceName, _ServiceMain },
249         { NULL, NULL }
250     };
251     if (m_bService && !::StartServiceCtrlDispatcher(st))
252     {
253         m_bService = FALSE;
254     }
255     if (m_bService == FALSE)
256         Run();
257 }
```



```
258
259 inline void CServiceModule::ServiceMain(DWORD /* dwArgc */, LPTSTR* /* lpszArgv */)
260 {
261     // Register the control request handler
262     m_status.dwCurrentState = SERVICE_START_PENDING;
263     m_hServiceStatus = RegisterServiceCtrlHandler(m_szServiceName, _Handler);
264     if (m_hServiceStatus == NULL)
265     {
266         CLogMsgEvent("Handler not installed").Post(_logAll);
267         return;
268     }
269     SetServiceStatus(SERVICE_START_PENDING);
270
271     m_status.dwWin32ExitCode = S_OK;
272     m_status.dwCheckPoint = 0;
273     m_status.dwWaitHint = 0;
274
275     // When the Run function returns, the service has stopped.
276     Run();
277
278     SetServiceStatus(SERVICE_STOPPED);
279 }
280
281 inline void CServiceModule::Handler(DWORD dwOpcode)
282 {
283     switch (dwOpcode)
284     {
285     case SERVICE_CONTROL_STOP:
286         SetServiceStatus(SERVICE_STOP_PENDING);
287         PostThreadMessage(dwThreadId, WM_QUIT, 0, 0);
288         break;
289     case SERVICE_CONTROL_PAUSE:
290         break;
291     case SERVICE_CONTROL_CONTINUE:
292         break;
293     case SERVICE_CONTROL_INTERROGATE:
294         break;
295     case SERVICE_CONTROL_SHUTDOWN:
296         break;
297     default:
298         CLogMsgEvent("Bad service request").Post(_logAll);
299     }
300 }
301
302 void WINAPI CServiceModule::_ServiceMain(DWORD dwArgc, LPTSTR* lpszArgv)
303 {
304     _Module.ServiceMain(dwArgc, lpszArgv);
305 }
306 void WINAPI CServiceModule::Handler(DWORD dwOpcode)
307 {
308     _Module.Handler(dwOpcode);
309 }
310
311 void CServiceModule::SetServiceStatus(DWORD dwState)
312 {
313     m_status.dwCurrentState = dwState;
314     ::SetServiceStatus(m_hServiceStatus, &m_status);
315 }
316
317 void CServiceModule::Run()
318 {
319     _Module.dwThreadId = GetCurrentThreadId();
320
321     HRESULT hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
322     if (FAILED(hr))
323     {
```

```

324     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_ERROR);
325     msg << "CoInitializeEx() failed. Error = {0x" << std::hex << hr << "}";
326     msg.Post(_logAll);
327     return;
328 }
329
330 // This provides a NULL DACL which will allow access to everyone.
331 CSecurityDescriptor sd;
332 sd.InitializeFromThreadToken();
333 hr = CoInitializeSecurity(sd, -1, NULL, NULL,
334     RPC_C_AUTHN_LEVEL_PKT, RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL);
335 _ASSERT(SUCCEEDED(hr));
336
337 hr = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_SERVER,
338     REGCLS_MULTIPLEUSE);
339 _ASSERT(SUCCEEDED(hr));
340
341 // MFC support
342 if (_theApp.InitApplication() == FALSE)
343 {
344     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_ERROR);
345     msg << "_theApp.InitApplication() failed";
346     msg.Post(_logAll);
347     return;
348 }
349
350 if (_theApp.InitInstance() == FALSE)
351 {
352     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_ERROR);
353     msg << "_theApp.InitInstance() failed";
354     msg.Post(_logAll);
355     _theApp.ExitInstance();
356     return;
357 }
358
359 // end MFC support
360
361
362 CLogMsgEvent("Service started").Post(_logAll);
363 if (m_bService)
364     SetServiceStatus(SERVICE_RUNNING);
365
366 _theApp.Run();
367
368 CLogMsgEvent("Service stopped").Post(_logAll);
369
370 _Module.RevokeClassObjects();
371 CoUninitialize();
372
373 //
374 }
375
376 //
377 //
378 extern "C" int WINAPI _twinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR
379     lpCmdLine,
380     int nShowCmd)
381 {
382     _logAll.AddLog(&_logEvents);
383     _logDebug.Enabled(false);
384     _logAll.AddLog(&_logFile);
385
386 #ifdef _DEBUG
387     _logEvents.EnableTranslation(true);
388     _logDebug.Enabled(true);

```

```
388     _logAll.AddLog(&_logDebug);
389 #endif
390
391     lpCmdLine = GetCommandLine(); //this line necessary for ATL MIN CRT
392     _Module.Init(ObjectMap, hInstance, IDS_SERVICENAME, &LIBID_LCKIOSKSERVERLib);
393     _Module.m_bService = TRUE;
394
395     TCHAR szTokens[] = _T("-/");
396
397     LPCTSTR lpszToken = FindOneOf(lpCmdLine, szTokens);
398     while (lpszToken != NULL)
399     {
400         if (lstrcmpi(lpszToken, _T("UnregServer"))==0)
401             return _Module.UnregisterServer();
402
403         // Register as Local Server
404         if (lstrcmpi(lpszToken, _T("RegServer"))==0)
405             return _Module.RegisterServer(TRUE, FALSE);
406
407         // Register as Service
408         if (lstrcmpi(lpszToken, _T("Service"))==0)
409             return _Module.RegisterServer(TRUE, TRUE);
410
411         // Initialize Configuration Registry Entries
412         // Initialize Configuration Registry Entries
413         if (lstrcmpi(lpszToken, _T("InitReg"))==0)
414         {
415             CRegistryKServer reg;
416             reg.buildInitial();
417             return 0;
418         }
419
420         lpszToken = FindOneOf(lpszToken, szTokens);
421     }
422
423     // Are we Service or Local Server
424     CRegKey keyAppID;
425     LONG lRes = keyAppID.Open(HKEY_CLASSES_ROOT, _T("AppID"), KEY_READ);
426     if (lRes != ERROR_SUCCESS)
427         return lRes;
428
429     CRegKey key;
430     lRes = key.Open(keyAppID, _T("(BF823564-E93B-11D3-B88C-CC792E000000)"), KEY_READ);
431     if (lRes != ERROR_SUCCESS)
432         return lRes;
433
434     TCHAR szValue[_MAX_PATH];
435     DWORD dwLen = _MAX_PATH;
436     lRes = key.QueryValue(szValue, _T("LocalService"), &dwLen);
437
438     _Module.m_bService = FALSE;
439     if (lRes == ERROR_SUCCESS)
440         _Module.m_bService = TRUE;
441
442     // AFX internal initialization
443     if (!AfxWinInit(hInstance, hPrevInstance, lpCmdLine, nShowCmd))
444         CLogMsgEvent(LCEV_GENERIC, SVRTY_ERROR, "AfxWinInit failed.").Post(_logAll);
445     else
446         _Module.Start();
447
448     // When we get here, the service has been stopped
449     return _Module.m_status.dwWin32ExitCode;
450 }
451
452
```

```
1 // threadReceiver.cpp : implementation file
2 //
3
4 #include "stdafx.h"
5 #include "LCKioskServer.h"
6 #include "threadReceiver.h"
7 #include "filenameDelimited.h"
8 #include "threadMain.h"
9 #include "registryKServer.h"
10 #include "hmlKioskCmds.h"
11 #include "Encryptor.h"
12 #include "exports.h"
13 #include "brokerWrap.h"
14
15
16 #ifdef _DEBUG
17 #define new DEBUG_NEW
18 #undef THIS_FILE
19 static char THIS_FILE[] = __FILE__;
20 #endif
21
22 ///////////////////////////////////////////////////////////////////
23 // CThreadReceiver
24
25 IMPLEMENT_DYNCREATE(CThreadReceiver, CThreadServer)
26
27 CThreadReceiver::CThreadReceiver()
28 {
29     useTimers(m_pnTimers, TIMER_MAX);
30 }
31
32 CThreadReceiver::~CThreadReceiver()
33 {
34 }
35
36 BOOL CThreadReceiver::InitInstance()
37 {
38     if (CThreadServer::InitInstance() == FALSE)
39         return FALSE;
40
41     m_fMaintenanceDone = false;
42     return TRUE;
43 }
44
45 int CThreadReceiver::ExitInstance()
46 {
47     // TODO: perform any per-thread cleanup here
48     return CThreadServer::ExitInstance();
49 }
50
51 BEGIN_MESSAGE_MAP(CThreadReceiver, CThreadServer)
52     //((AFX_MSG_MAP(CThreadReceiver)
53     //    ON_THREAD_MESSAGE(WMUSER_START, onStart)
54     //))AFX_MSG_MAP
55     ON_THREAD_MESSAGE(WMUSER_START, onStart)
56 END_MESSAGE_MAP()
57
58 ///////////////////////////////////////////////////////////////////
59 // CThreadReceiver message handlers
60
61 LRESULT CThreadReceiver::onStart(WPARAM wParam, LPARAM lParam)
62 {
63     CThreadServer::onStart(wParam, lParam);
64     setTimer(TIMER_RECEIVE, 2000);
65
66     // todo: testing
```

```
67     setTimer(TIMER_CHECK_TIME, 1000);
68     //setTimer(TIMER_CHECK_TIME, INTERVAL_CHECK_TIME);
69
70     setTimer(TIMER_MOVE_FILES, INTERVAL_TIMER_MOVE_FILES);
71
72     return FALSE;
73 }
74
75 void CThreadReceiver::onTimerIndex(int nIndex)
76 {
77     killTimer(nIndex);
78
79     SYSTEMTIME st;
80     GetLocalTime(&st);
81     SystemTimeToVariantTime(&st, &m_dateNow);
82
83     switch (nIndex)
84     {
85     case TIMER_RECEIVE:
86         onTimerReceiveFiles();
87         break;
88     case TIMER_CHECK_TIME:
89         onTimerCheckTime();
90         break;
91     case TIMER_MOVE_FILES:
92         onTimerMoveFiles();
93         break;
94     default:
95         break;
96     }
97
98     return;
99 }
100
101 void CThreadReceiver::onTimerCheckTime()
102 {
103     SYSTEMTIME st;
104     GetLocalTime(&st);
105     COleDateTime odtNow;
106     odtNow.SetTime(st.wHour, st.wMinute, st.wSecond);
107
108     COleDateTimeSpan odtSpan = odtNow - _theApp.m_registry->m_odtEndOfDay;
109     int nMinutes = odtSpan.GetTotalMinutes();
110     if (nMinutes > 0 && nMinutes < 60)
111     {
112         if (!m_fMaintenanceDone)
113         {
114             m_fMaintenanceDone = true;
115             deleteBackupFiles();
116             checkLateKiosks();
117             buildExports();
118             buildUserExports();
119         }
120     }
121     else
122         m_fMaintenanceDone = false;
123
124     setTimer(TIMER_CHECK_TIME, INTERVAL_CHECK_TIME);
125 }
126
127 void CThreadReceiver::onTimerReceiveFiles()
128 {
129     CFileNameKiosk fnKiosk;
130     CFileFind ffLocal;
131     string strSearchFileName;
132     CString cstrFileName;
```

```

133     string strKioskId;
134     string strWork;
135     string strBackupFile;
136     CFile file;
137     string strXmlCmd;
138     string strXmlResult;
139     string strXmlError;
140
141     // set up wild card search name
142     fnKiosk.set("direction", "U");
143     fnKiosk.set("kiosk_id", "");
144     fnKiosk.set("date", "");
145     fnKiosk.setExtension("XML");
146     fnKiosk.getFullName(strSearchFileName);
147
148     // set current directory to the xfer directory
149     _chdrive(_theApp.m_pregistry->m_lKioskDrive);
150     string strDirectory = _theApp.m_pregistry->m_strKioskDirectory;
151     strDirectory += "Xfer\\";
152     _chdir(strDirectory.c_str());
153
154     // look at each file in the xfer directory that matches wild card search
155     bool fFileFound = ffLocal.FindFile(strSearchFileName.c_str()) != FALSE;
156     while (fFileFound)
157     {
158         fFileFound = ffLocal.FindNextFile() != FALSE;
159         cstrFileName = ffLocal.GetFileName();
160
161         // parse filename and get out the kiosk id
162         fnKiosk.setFullName(cstrFileName);
163         fnKiosk.get("kiosk_id", strWork);
164         strKioskId = strWork.substr(1, strWork.size() - 1); // drop K prefix
165
166         // file errors can occur if we hit a file that is being xmitted, error is a
167         warning
168         CFileException exFile;
169         if (file.Open(cstrFileName, CFile::modeRead | CFile::shareExclusive, &exFile)
170         == FALSE)
171         {
172             char pszError [256];
173             *pszError = 0;
174             exFile.GetErrorMessage((LPSTR) pszError, 255);
175             CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
176             msg << "CThreadReceiver unable to open transaction file {";
177             msg << (const char *) cstrFileName << "} for storage. Error = {";
178             msg << pszError << "}. Cause = {" << exFile.m_cause << "}";
179             msg.Post(_logAll);
180             continue;
181         }
182
183         // read in the contents of the file. error is a warning.
184         long lFileSize = file.GetLength();
185         unsigned char * pBuff = new unsigned char [lFileSize + 1];
186         pBuff[lFileSize] = 0;
187         if (file.Read(pBuff, lFileSize) != lFileSize)
188         {
189             CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
190             msg << "CThreadReceiver unable to read transaction file {";
191             msg << (const char *) cstrFileName << "} for storage.";
192             msg.Post(_logAll);
193             file.Close();
194             continue;
195         }
196         file.Close();
197
198         // build xml command

```

```
197 Cxdoc uptKioskTrans docXmlCmd;
198 docXmlCmd.setItemText("kiosk id", strKioskId.c_str());
199 docXmlCmd.setItemText("file_name", cstrFileName);
200
201 // decrypt incoming data
202 CEncryptor encrypt;
203 string strTransXml;
204 encrypt.Decrypt((LPCTSTR) pBuff, NULL, strTransXml);
205 delete [] pBuff;
206
207 // parse incoming data
208 CXmlDocument docXmlTrans(strTransXml.c_str());
209 if (!docXmlTrans.IsReady())
210 {
211     string strError;
212     docXmlTrans.GetParserError(strError);
213
214     // event log message
215     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
216     msg << "CThreadReceiver unable to parse transaction file [";
217     msg << (const char *) cstrFileName << "]. XML Parser error - [";
218     msg << strError << "];";
219     msg.Post(_logAll);
220
221     // alert personnel message
222     stringstream strmError;
223     strmError << "The transaction file [" << (const char *) cstrFileName;
224     strmError << "] could not be parsed. The error is [" << strError << "]. ";
225     strmError << "You are listed as support personnel for this kiosk [" <<
226     strKioskId;
227     strmError << "]. Please investigate this problem ASAP.";
228
229     Cxdoc_setKioskAlert docAlert;
230     CXmlElement elParm;
231
232     docAlert.GetItem("parm", &elParm, Cxdoc_setKioskAlert::PARAM_kiosk_id);
233     elParm.SetText(strKioskId.c_str());
234     docAlert.GetItem("parm", &elParm, Cxdoc_setKioskAlert::PARAM_error_level);
235     elParm.SetText(2);
236     docAlert.GetItem("parm", &elParm, Cxdoc_setKioskAlert::PARAM_subject);
237     elParm.SetText("Kiosk transaction file parse error");
238     docAlert.GetItem("parm", &elParm, Cxdoc_setKioskAlert::PARAM_body);
239     elParm.SetText(strmError.str().c_str());
240     docAlert.GetXML(strXmlCmd);
241
242     CBrokerWrap broker(m_spXmlCmds);
243     if (!broker.ExecXml(strXmlCmd.c_str(), strXmlResult, strXmlError))
244     {
245         msg.Clear();
246         msg << "CThreadReceiver unable to issue Kiosk alert.";
247         msg.Post(_logAll);
248     }
249     continue;
250 }
251
252 CXmlElement elTrans;
253 CXmlElement elData;
254
255 docXmlTrans.GetRoot(&elTrans);
256 docXmlCmd.GetItem("data", &elData);
257
258 elData.AddChild(&elTrans);
259
260 docXmlCmd.GetXML(strXmlCmd);
261
```

```
262 // put it to the broker
263 CBrokerWrap broker(m_spXmlCmds);
264 if (!broker.execXml(strXmlCmd.c_str(), strXmlResult, strXmlError))
265 {
266     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
267     msg << "CThreadReceiver unable to store Kiosk transaction file [";
268     msg << (const char *) cstrFileName << "].";
269     msg.Post(_logAll);
270     continue;
271 }
272
273 // move to backup directory
274 strBackupFile = "..\\Backup\\";
275 strBackupFile += (const char *) cstrFileName;
276 if (rename(cstrFileName, strBackupFile.c_str()))
277 {
278     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
279     msg << "Unable to move file [" << (const char *) cstrFileName << "] to thew";
280     msg << "backup directory. errno = [" << errno << "]);";
281     msg.Post(_logAll);
282 }
283
284
285 setTimer(TIMER_RECEIVE, _theApp.m_pregistry->m_lReceiveFreqMs);
286 return;
287 }
288
289 void CThreadReceiver::onTimerMoveFiles()
290 {
291     CFileFind ffLocal;
292     string strWorkDir = _theApp.m_pregistry->m_strKioskDirectory + "Work\\";
293     string strXferDir = _theApp.m_pregistry->m_strKioskDirectory + "Xfer\\";
294
295     string strSrcFile;
296     string strDestFile;
297     CString cstrFileName;
298
299     // move files out to the xfer directory
300     string strFileFind = strWorkDir + ".*.*";
301     BOOL fFileFound = ffLocal.FindFile(strFileFind.c_str());
302     while (fFileFound)
303     {
304         fFileFound = ffLocal.FindNextFile();
305         if (ffLocal.IsDirectory())
306             continue;
307         cstrFileName = ffLocal.GetFileName();
308         strSrcFile = (LPCSTR) cstrFileName;
309         strSrcFile.insert(0, strWorkDir);
310         strDestFile = (LPCSTR) cstrFileName;
311         strDestFile.insert(0, strXferDir);
312
313         remove(strDestFile.c_str());
314         rename(strSrcFile.c_str(), strDestFile.c_str());
315     }
316
317     // remove files marked as retrieved
318     strFileFind = strXferDir + ".*.*x";
319     fFileFound = ffLocal.FindFile(strFileFind.c_str());
320     while (fFileFound)
321     {
322         fFileFound = ffLocal.FindNextFile();
323         if (ffLocal.IsDirectory())
324             continue;
325         cstrFileName = ffLocal.GetFilePath();
326         remove(cstrFileName);
327     }
328 }
```



```
327     }
328
329     setTimer(TIMER_MOVE_FILES, INTERVAL_TIMER_MOVE_FILES);
330
331     return;
332 }
333
334 bool CThreadReceiver::deleteBackupFiles()
335 {
336     bool fSuccess = true;
337
338     // delete backup file on disk
339     string strFileName;
340     string strSearchName("*.");
341     string strBackupDirectory = _theApp.m_pregistry->m_strKioskDirectory;
342     strBackupDirectory += "Backup\\";
343     strSearchName.insert(0, strBackupDirectory);
344
345     // get current time - retentions days
346     SYSTEMTIME stime;
347     GetLocalTime(&stime);
348     COleDateTime odtKeep(stime);
349     COleDateTimeSpan spanTime(_theApp.m_pregistry->m_lBackupRetentionDays, 0, 0, 0);
350     odtKeep -= spanTime;
351
352     // delete all file whose time is less than current time minus retention days
353     CFileFind ffLocal;
354     bool fFileFound = ffLocal.FindFile(strSearchName.c_str()) != FALSE;
355     while (fFileFound)
356     {
357         fFileFound = ffLocal.FindNextFile() != FALSE;
358         if (ffLocal.IsDots())
359             continue;
360
361         FILETIME ftime;
362         ffLocal.GetCreationTime(&ftime);
363         COleDateTime odtFile(ftime);
364         if (odtFile < odtKeep)
365         {
366             CString cstrFileName = ffLocal.GetFilePath();
367             remove(cstrFileName);
368         }
369     }
370
371     // remove old daily transactions from database
372     odtKeep.SetDate(stime.wYear, stime.wMonth, stime.wDay);
373     spanTime.SetDateTimeSpan(_theApp.m_pregistry->m_lTransRetentionDays, 0, 0, 0);
374     odtKeep -= spanTime;
375
376     CXmlElement elDate;
377     Cxdoc_deleteKioskTrans xdoc_deleteKioskTrans;
378     xdoc_deleteKioskTrans.getItem("parm", &elDate);
379     elDate.setText((LPCSTR) odtKeep.Format("%Y-%m-%d"));
380     string strXmlCmd;
381     string strXmlResult;
382     string strXmlError;
383     xdoc_deleteKioskTrans.getXML(strXmlCmd);
384     CBrokerWrap broker(m_spXmlCmds);
385     fSuccess = broker.execXml(strXmlCmd.c_str(), strXmlResult, strXmlError);
386
387     return fSuccess;
388 }
389
390 bool CThreadReceiver::checkLateKiosks()
391 {
392     Cxdoc_checkLateKiosks docCmd;
```

```
393
394     CXMLElement elParm;
395
396     docCmd.getItem("parm", &elParm);
397     elParm.setText(_theApp.m_pregistry->m_lHoursKioskAbsent);
398
399     string strXmlCmd;
400     string strXmlResult;
401     string strXmlError;
402
403     docCmd.getXML(strXmlCmd);
404
405     CBrokerWrap broker(m_spXmlCmds);
406     if (!broker.execXml(strXmlCmd.c_str(), strXmlResult, strXmlError))
407     {
408         CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
409         msg << "CThreadReceiver unable to checkLateKiosks().";
410         msg.Post(_logAll);
411     }
412
413     return true;
414 }
415
416 bool CThreadReceiver::buildExports()
417 {
418     CExportKiosks ek(m_spXmlCmds);
419     string strDir = _theApp.m_pregistry->m_strKioskDirectory + "Xfer\\";
420     ek.setFinalDir(strDir.c_str());
421     strDir = _theApp.m_pregistry->m_strKioskDirectory + "Work\\Kiosks\\";
422     ek.setWorkDir(strDir.c_str());
423     return ek.buildDailyExport();
424 }
425
426 bool CThreadReceiver::buildUserExports()
427 {
428     string strDir;
429
430     CExportLCUsers eu(m_spXmlCmds);
431     strDir = _theApp.m_pregistry->m_strKioskDirectory + "Work\\";
432     eu.setFinalDir(strDir.c_str());
433     strDir += "LCUsers\\";
434     eu.setWorkDir(strDir.c_str());
435     eu.setPacketSize(_theApp.m_pregistry->m_lLCUsersPerPacket);
436
437     // build the big kahonee
438     COleDateTime odt;
439     odt.SetDate(1990, 12, 29);
440     eu.buildDailyExport((DATE) odt, NULL, true);
441
442     // build the daily
443     odt = COleDateTime::GetCurrentTime();
444     eu.buildDailyExport((DATE) odt, NULL, false);
445
446     return true;
447 }
448
449
```

```

1 // wndMonitorISP.cpp : implementation file
2 //
3
4 #include "stdafx.h"
5 #include "lckioskclient.h"
6 #include "wndMonitorISP.h"
7 #include "filenameDelimited.h"
8 #include "zipUtil.h"
9 #include "Encryptor.h"
10
11 #ifdef _DEBUG
12 #define new DEBUG_NEW
13 #undef THIS_FILE
14 static char THIS_FILE[] = __FILE__;
15 #endif
16
17 /*
18
19 This window acts like the following state machine. A state change results in
20 either a posted message or a set timer whose event will process that state.
21
22 state          | new state          | condition
23 -----|-----|-----
24 ST_CHECKTIME   | ST_CHECKTIME       | when it is not time to do anything
25               | ST_CONNECTING      | when it is time to process
26 -----|-----|-----
27 ST_CONNECTING  | ST_RETRY           | when the connect fails
28               | ST_EXCHANGE        | when the connect succeeds
29               | ST_CHECKTIME       | when connect fails and retries are exhausted
30 -----|-----|-----
31 ST_RETRY       | ST_CONNECTING      |
32 -----|-----|-----
33 ST_EXCHANGE    | ST_APPLY           | when the data exchange succeeds
34               | ST_RETRY           | when the data exchange fails
35               | ST_CHECKTIME       | when the data exchange fails and retries are exhausted
36 -----|-----|-----
37 ST_APPLY       | ST_CHECKTIME       | when apply updates succeeds
38               | ST_CHECKTIME       | when apply updates fails
39 */
40
41 UINT CWndMonitorISP::m_nRasDialMsg = 0;
42
43 //////////////////////////////////////
44 // CWndMonitorISP
45
46 CWndMonitorISP::CWndMonitorISP()
47 {
48     m_fUseRas = true;
49     m_pdialer = NULL;
50     m_pdialerRas = NULL;
51     m_pdialerWinInet = NULL;
52     m_fExchangeDone = false;
53     m_fMaintenanceDone = false;
54 }
55
56 CWndMonitorISP::~CWndMonitorISP()
57 {
58     if (m_pdialer != NULL)
59         delete m_pdialer;
60 }
61
62 BEGIN_MESSAGE_MAP(CWndMonitorISP, CWndMonitor)
63     //{{AFX_MSG_MAP(CWndMonitorISP)
64     //{{AFX_MSG_MAP(CWndMonitorISP)
65     //{{AFX_MSG_MAP(CWndMonitorISP)

```

```
66  //))AFX MSG_MAP
67  ON_MESSAGE(WMUSER_CONNECTED, onDialConnect)
68  ON_MESSAGE(WMUSER_CHECKTIME, onTimerCheckTime)
69  ON_MESSAGE(WMUSER_EXCHANGE, onExchange)
70  ON_MESSAGE(WMUSER_APPLY, onApply)
71  ON_MESSAGE(WMUSER_CONNECT, onTryConnect)
72  ON_REGISTERED_MESSAGE(m_nRasDialMsg, onRasDialReport)
73 END_MESSAGE_MAP()
74
75
76 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
77 // CWndMonitorISP message handlers
78
79 int CWndMonitorISP::OnCreate(LPCREATESTRUCT lpCreateStruct)
80 {
81     if (CWndMonitorISP::OnCreate(lpCreateStruct) == -1)
82         return -1;
83
84     buildDirectoryStructure();
85
86     if (m_fUseRas)
87     {
88         m_pdialerRas = new CDialerRAS(m_registry.m_strPhoneBookEntry.c_str());
89         m_nRasDialMsg = m_pdialerRas->getNotificationMessId();
90         m_pdialer = m_pdialerRas;
91     }
92     else
93     {
94         m_pdialerWinInet = new CDialerWinInet();
95         m_pdialer = m_pdialerWinInet;
96     }
97
98     m_pdialer->m_hwndOwner = m_hWnd;
99
100     m_ePreviousState = ST_NONE;
101     setState(ST_CHECKTIME);
102     PostMessage(WMUSER_CHECKTIME);
103     return 0;
104 }
105
106 void CWndMonitorISP::OnTimer(UINT nIDEvent)
107 {
108     KillTimer(nIDEvent);
109
110     switch(nIDEvent)
111     {
112     case TIMER_CHECKTIME:
113         onTimerCheckTime(0, 0);
114         break;
115     case TIMER_RETRY:
116         onTimerRetry();
117         break;
118     default:
119         break;
120     }
121 }
122
123 LRESULT CWndMonitorISP::onTimerCheckTime(WPARAM wParam, LPARAM lParam)
124 {
125     COleDateTime odtNow = COleDateTime::GetCurrentTime();
126
127     // check to see if it is end of day. If it is, then do maintenance
128     COleDateTimeSpan odtNowSpan = odtNow - m_registry.m_odtEndOfDay;
129     int nMinutes = odtNowSpan.GetTotalMinutes();
130
131     if (nMinutes > 0 && nMinutes < 60)
```

```
132 {
133     if (!m_fMaintenanceDone)
134     {
135         deleteOldBackups();
136         cleanChartersDirectory();
137         return true;
138     }
139 }
140 else
141     m_fMaintenanceDone = false;
142
143 // check to see if we should do file exchange
144 odtNow = odtNow - m_registry.m_odtTimeOfExchange;
145 nminutes = odtNow.GetTotalMinutes();
146
147 if (nminutes >= 0)
148 {
149     // start the exchange
150     if (m_registry.m_lConnectRetries)
151         m_nRetryCount = m_registry.m_lConnectRetries;
152     else
153         m_nRetryCount = RETRY_FOREVER;
154
155     setState(ST_CONNECTING);
156     PostMessage(WMUSER_CONNECT);
157
158     // set up the next exchange time
159     COleDateTimeSpan odtMinutes(0, 0, m_registry.m_lExchangeFreqMins, 0);
160     m_registry.m_odtTimeOfExchange = odtNow + odtMinutes;
161 }
162 else
163 {
164     setState(ST_CHECKTIME);
165     SetTimer(TIMER_CHECKTIME, INTERVAL_CHECKTIME, NULL);
166 }
167
168 return 0;
169 }
170
171 bool CWndMonitorISP::onTimerRetry()
172 {
173     if (m_nRetryCount > 0)
174         m_nRetryCount--;
175
176     CLogMsgEvent(LCEV_GENERIC, SVRTY_INFO, "Retrying connect").Post(_logAll);
177
178     setState(ST_CONNECTING);
179     PostMessage(WMUSER_CONNECT);
180
181     return true;
182 }
183
184 HRESULT CWndMonitorISP::onTryConnect(WPARAM wParam, LPARAM lParam)
185 {
186     if (m_registry.m_lNoDial != 0)
187     {
188         setState(ST_EXCHANGING);
189         PostMessage(WMUSER_EXCHANGE);
190     }
191     else
192         m_pdialer->connect();
193
194     return 0;
195 }
196
197 HRESULT CWndMonitorISP::onExchange(WPARAM wParam, LPARAM lParam) [Claim 1b, 4a, 5a, 5b, 5c, 4
```

```
    7b,9b,10a]
198 {
199     CLogMsgEvent(LCEV_GENERIC, SVRTY_INFO, "Exchanging files").Post(_logAll);
200
201     bool fSuccess = exchangeFiles();
202
203     m_pdialer->disconnect();
204
205     if (fSuccess)
206     {
207         m_fMaintenanceDone = true;
208         setState(ST_APPLYING);
209         PostMessage(WMUSER_APPLY);
210     }
211     else
212     {
213         if (m_nRetryCount > 0 || m_nRetryCount == RETRY_FOREVER)
214         {
215             setState(ST_RETRYING);
216             SetTimer(TIMER_RETRY, m_registry.m_lRetryIntervalSecs * 1000, NULL);
217         }
218         else
219         {
220             setState(ST_CHECKTIME);
221             SetTimer(TIMER_CHECKTIME, INTERVAL_CHECKTIME, 0);
222         }
223     }
224
225     return fSuccess ? TRUE : FALSE;
226 }
227
228 LRESULT CWndMonitorISP::onRasDialReport(WPARAM wRasConnState, LPARAM dwError)
229 {
230     // if doing async Ras, logNotification must be called to recieve WMUSER_CONNECTED ✓
231     message
232     m_pdialerRas->logNotification(wRasConnState, dwError);
233     return 0;
234 }
235 LRESULT CWndMonitorISP::onDialConnect(WPARAM wParam, LPARAM dwError)
236 {
237     if (dwError)
238     {
239         m_pdialer->disconnect();
240         if (m_nRetryCount > 0 || m_nRetryCount == RETRY_FOREVER)
241         {
242             setState(ST_RETRYING);
243             SetTimer(TIMER_RETRY, m_registry.m_lRetryIntervalSecs * 1000, NULL);
244         }
245         else
246         {
247             setState(ST_CHECKTIME);
248             SetTimer(TIMER_CHECKTIME, INTERVAL_CHECKTIME, 0);
249         }
250     }
251     else
252     {
253         setState(ST_EXCHANGING);
254         PostMessage(WMUSER_EXCHANGE);
255     }
256
257     return 0;
258 }
259
260 bool CWndMonitorISP::_funcSortFileName(string & str1, string & str2)
261 {
```

```
262     return str1.compare(str2) < 0;
263 }
264
265 bool CWndMonitorISP::applyLCUsersUpdates() [Claim 1f,1g]
266 {
267     string strSearchName;
268     string strFileName;
269     string strBackupName;
270     CString cstrZipFileName;
271     CString cstrXmlFileName;
272
273     CLogMsgEvent(LCEV_GENERIC, SVRTY_INFO, "Applying LCUsers.").Post(_logAll);
274
275     bool fSuccess = true;
276
277     try
278     {
279         string strProcDir = m_strProcDir + "LCUsers\\";
280
281         strSearchName = "D_LCUsers*.zip";
282         strSearchName.insert(0, m_strProcDir);
283
284         CFileFind ffZips;
285         BOOL fZipFound = ffZips.FindFile(strSearchName.c_str());
286         while (fZipFound)
287         {
288             fZipFound = ffZips.FindNextFile();
289             cstrZipFileName = ffZips.GetFilePath();
290             clearDirectory(strProcDir.c_str());
291             m_zipper.unzipFile(cstrZipFileName, strProcDir.c_str());
292
293             strSearchName = strProcDir + "*.xml";
294             CFileFind ffXmIs;
295             BOOL fXmlFound = ffXmIs.FindFile(strSearchName.c_str());
296             while (fXmlFound)
297             {
298                 fXmlFound = ffXmIs.FindNextFile();
299                 cstrXmlFileName = ffXmIs.GetFilePath();
300                 m_spKCData->importLCUsers(_variant_t(cstrXmlFileName));
301                 remove(cstrXmlFileName);
302             }
303
304             strBackupName = m_strBackupDir + (LPCTSTR) ffZips.GetFileName();
305             rename(cstrZipFileName, strBackupName.c_str());
306         }
307     }
308     catch(_com_error & e)
309     {
310         CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
311         msg << "Error during applyLCUsersUpdates(). Error = [";
312         msg.appendError(e);
313         msg << "].";
314         msg.Post(_logAll);
315         fSuccess = false;
316     }
317     catch(CException * pE)
318     {
319         CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
320         char pszErrorMessage [256];
321         pE->GetErrorMessage(pszErrorMessage, 256);
322         msgError << "Error during applyLCUsersUpdates(). Error = [" << pszErrorMessage
323         << "].";
324         pE->Delete();
325         fSuccess = false;
326     }
```

```
327     catch(...)
328     {
329         CLogMsgEvent(LCEV_GENERIC, SVRTY_WARNING, "Unknown exception during
        applyLCUsersUpdates().").Post(_logAll);
330         fSuccess = false;
331     }
332
333
334     return fSuccess;
335 }
336
337 bool CWndMonitorISP::applyAppUpdates()
338 {
339     string      strSearchName;
340     string      strFileName;
341     string      strBackupName;
342     CString     cstrFileName;
343
344     CLogMsgEvent    msgInfo(LCEV_GENERIC, SVRTY_INFO);
345
346     msgInfo << "Applying application updates.";
347     msgInfo.Post(_logAll);
348
349     bool fSuccess = true;
350
351     try
352     {
353         vector<string> collFileNamesAll;
354         vector<string> collFileNamesById;
355
356         //////////////////////////////////////
357         // find application zip files
358         //////////////////////////////////////
359         // build wild card file name for KALL search
360         CFileNameKiosk fnKiosk;
361         fnKiosk.set("direction", "D");
362         fnKiosk.set("kiosk_id", "KALL");
363         fnKiosk.set("date", "*");
364         fnKiosk.setExtension("zip");
365         fnKiosk.getFullName(strSearchName);
366         strSearchName.insert(0, m_strProcDir);
367
368         // collect all KALL file name and sort them by date
369         CFileFind ffLocal;
370         BOOL fFileFound = ffLocal.FindFile(strSearchName.c_str());
371         while (fFileFound)
372         {
373             fFileFound = ffLocal.FindNextFile();
374             cstrFileName = ffLocal.GetFileName();
375             collFileNamesAll.push_back((const char *) cstrFileName);
376         }
377         sort(collFileNamesAll.begin(), collFileNamesAll.end(), _funcSortFileName);
378
379         // build wild card file name for K9999 search
380         fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
381         fnKiosk.getFullName(strSearchName);
382         strSearchName.insert(0, m_strProcDir);
383
384         // collect all K9999 file names and sort them by date
385         fFileFound = ffLocal.FindFile(strSearchName.c_str());
386         while (fFileFound)
387         {
388             fFileFound = ffLocal.FindNextFile();
389             cstrFileName = ffLocal.GetFileName();
390             collFileNamesById.push_back((const char *) cstrFileName);
391         }
    }
```



```
392     sort(collFileNamesById.begin(), collFileNamesById.end(), _funcSortFileName);
393
394     // append K9999 filenames to the end of KALL filenames
395     collFileNamesAll.insert(collFileNamesAll.end(), collFileNamesById.begin(),
396                             collFileNamesById.end());
397
398     //////////////////////////////////////
399     // apply application zip files
400     //////////////////////////////////////
401     vector<string>::iterator it;
402     for (it = collFileNamesAll.begin(); it != collFileNamesAll.end(); it++)
403     {
404         strFileName = m_strProcDir + *it;
405         if (!m_zipper.unzipFile(strFileName.c_str(), m_registry.m_strAppDirectory.
406                                c_str(),
407                                CZipUtil::ZF_OverWrite | CZipUtil::ZF_UseDirectoryNames))
408         {
409             CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
410             msgError << "Unable to apply [" << *it << "]";
411             msgError.Post(_logAll);
412         }
413         else
414         {
415             msgInfo.clear();
416             msgInfo << "Applied update file [" << *it << "]";
417             msgInfo.Post(_logAll);
418         }
419         strBackupName = m_strBackupDir + *it;
420         strFileName = m_strProcDir + *it;
421         if (rename(strFileName.c_str(), strBackupName.c_str()))
422         {
423             CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
424             msgError << "Unable to move [" << *it << "] to the backup directory
425             after applying.";
426             msgError << " errno - " << errno;
427             msgError.Post(_logAll);
428         }
429     }
430     catch(_com_error & e)
431     {
432         CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
433         msg << "Error during applyAppUpdates(). Error = ";
434         msg.appendError(e);
435         msg << ".";
436         msg.Post(_logAll);
437         fSuccess = false;
438     }
439     catch(CException * pE)
440     {
441         CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
442         char pszErrorMessage [256];
443         pE->GetErrorMessage(pszErrorMessage, 256);
444         msgError << "Error during applyAppUpdates(). Error = [" << pszErrorMessage <<
445         "]";
446         pE->Delete();
447         fSuccess = false;
448     }
449     catch(...)
450     {
451         CLogMsgEvent(LCEV_GENERIC, SVRTY_WARNING, "Unknown exception during
452         applyAppUpdates().").Post(_logAll);
453         fSuccess = false;
454     }
455 }
```

```
453     return fSuccess;
454 }
455
456 bool CWndMonitorISP::applyDBUpdates()
457 {
458     string strSearchName;
459     string strFileName;
460     string strBackupName;
461     CString cstrFileName;
462
463     bool fSuccess = true;
464
465     CLogMsgEvent(LCEV_GENERIC, SVRTY_INFO, "Applying database updates.").Post(_logAll);
466
467     try
468     {
469         CFileNameKiosk fnKiosk;
470         fnKiosk.set("direction", "D");
471         fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
472         fnKiosk.set("date", "");
473         fnKiosk.setExtension("xml");
474         fnKiosk.getFullName(strSearchName);
475         strSearchName.insert(0, m_strProcDir);
476
477         CFileFind ffLocal;
478         BOOL fFileFound = ffLocal.FindFile(strSearchName.c_str());
479         while (fFileFound)
480         {
481             fFileFound = ffLocal.FindNextFile();
482             cstrFileName = ffLocal.GetFileName();
483             strFileName = (LPCSTR) cstrFileName;
484             strFileName.insert(0, m_strProcDir);
485             m_spKCDATA->importData(_variant_t((LPCSTR) strFileName.c_str()));
486
487             strBackupName = m_strBackupDir + (LPCSTR) cstrFileName;
488             rename(strFileName.c_str(), strBackupName.c_str());
489         }
490     }
491     catch(_com_error & e)
492     {
493         CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
494         msg << "Error during applyDBUpdates(). Error = {";
495         msg.appendError(e);
496         msg << "}. ";
497         msg.Post(_logAll);
498         fSuccess = false;
499     }
500     catch(CException * pE)
501     {
502         CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
503         char pszErrorMessage [256];
504         pE->GetErrorMessage(pszErrorMessage, 256);
505         msgError << "Error during applyDBUpdates(). Error = {" << pszErrorMessage <<
506         "}]";
507         pE->Delete();
508         fSuccess = false;
509     }
510     catch(...)
511     {
512         CLogMsgEvent(LCEV_GENERIC, SVRTY_WARNING, "Unknown exception during
513         applyDBUpdates().").Post(_logAll);
514         fSuccess = false;
515     }
516
517     return true;
518 }
```

```
516 }
517
518 HRESULT CWndMonitorISP::onApply(WPARAM, LPARAM)
519 {
520     bool fSuccess = true;
521
522     try
523     {
524         HRESULT hr = m_spKCDATA.CreateInstance(__uuidof(KCDATA));
525         if (FAILED(hr))
526         {
527             CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
528             msg << "Unable to instantiate IKCDATA. Error = [0x" << std::hex << hr <<
529             ".]";
530             msg << " Database updates were not applied.";
531             msg.Post(_logAll);
532             throw E_FAIL;
533         }
534         m_spKCDATA->open();
535
536         if (!m_zipper.CreateInstance())
537             throw E_FAIL;
538
539         m_strBackupDir = m_registry.m_strLocalDirectory + "Backup\\";
540         m_strProcDir = m_registry.m_strLocalDirectory + "Process\\";
541
542         applyDBUpdates();
543         applyLCUsersUpdates();
544         applyAppUpdates();
545
546         m_spKCDATA->close();
547     }
548     catch(_com_error & e)
549     {
550         CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
551         msg << "Error during onApply(). Error = [";
552         msg.appendError(e);
553         msg << "].";
554         msg.Post(_logAll);
555         fSuccess = false;
556     }
557     catch(HRESULT hrError)
558     {
559         hrError;
560         fSuccess = false;
561     }
562
563     // done, go back into a wait state till next maintenance run
564     setState(ST_CHECKTIME);
565     SetTimer(TIMER_CHECKTIME, INTERVAL_CHECKTIME, 0);
566
567     return fSuccess ? TRUE : FALSE;
568 }
569
570 void CWndMonitorISP::clearDirectory(const char * pszDir)
571 {
572     string strSearchName = pszDir;
573     strSearchName += ".*.*";
574
575     CFileFind ffLocal;
576     BOOL fFileFound = ffLocal.FindFile(strSearchName.c_str());
577     while (fFileFound)
578     {
579         fFileFound = ffLocal.FindNextFile();
580         if (ffLocal.IsDirectory())
```

```
581         continue;
582         CString cstrFileName = ffLocal.GetFilePath();
583         remove(cstrFileName);
584     }
585
586     return;
587 }
588
589 bool CWndMonitorISP::cleanChartersDirectory()
590 {
591     if (m_registry.m_strChartDir.size() == 0)
592         return true;
593
594     FILETIME ft;
595     COleDateTime odtFile;
596     COleDateTime odtYesterday = COleDateTime::GetCurrentTime();
597     COleDateTimeSpan spanDay(1, 0, 0, 0);
598     odtYesterday -= spanDay;
599
600     CString cstrFileName;
601     string strSearchName = m_registry.m_strChartDir + ".*";
602
603     CFileFind ffCharts;
604
605     BOOL fFileFound = ffCharts.FindFile(strSearchName.c_str());
606     while (fFileFound)
607     {
608         fFileFound = ffCharts.FindNextFile();
609         if (ffCharts.IsDirectory())
610             continue;
611         ffCharts.GetLastWriteTime(&ft);
612         odtFile = ft;
613         if (odtFile < odtYesterday)
614         {
615             cstrFileName = ffCharts.GetFilePath();
616             remove(cstrFileName);
617         }
618     }
619
620     return true;
621 }
622
623 bool CWndMonitorISP::deleteOldBackups()
624 {
625     bool fSuccess = true;
626
627     // make the Backup directory current
628     string strBackupDir = m_registry.m_strLocalDirectory + "Backup\\";
629     chdrive(m_registry.m_lLocalDrive);
630     _chdir(strBackupDir.c_str());
631
632     CFileFind ffBackup;
633     CString cstrFileName;
634     CFileNameKiosk fnKiosk;
635     COleDateTime odtDeleteDate;
636     COleDateTimeSpan spanRetentionPeriod;
637
638     // set up the delete date by taking todays date and subtracting backup retention
639     days spanRetentionPeriod.SetDateTimeSpan(m_registry.m_lBackupRetentionDays, 0, 0, 0);
640     SYSTEMTIME st;
641     GetLocalTime(&st);
642     odtDeleteDate.SetDate(st.wYear, st.wMonth, st.wDay);
643     odtDeleteDate -= spanRetentionPeriod;
644
645     // set up wild card search
```

```

646     fnKiosk.set("direction", "");
647     fnKiosk.set("kiosk_id", "");
648     fnKiosk.set("date", "");
649     fnKiosk.setExtension("");
650     string strSearch;
651     fnKiosk.getFullName(strSearch);
652
653     // retrieve list of files in backup directory
654     bool fFileFound = ffBackup.FindFile(strSearch.c_str()) != FALSE;
655     while (fFileFound)
656     {
657         fFileFound = ffBackup.FindNextFile() != FALSE;
658         CString cstrFileName = ffBackup.GetFileName();
659         fnKiosk.setFullName(cstrFileName);
660
661         // retrieve the date portion of the file name
662         DATE dateFile;
663         fnKiosk.get("date", dateFile);
664         COleDateTime odtFile(dateFile);
665
666         // if the date falls before now minus retention period, then delete it
667         if (odtFile < odtDeleteDate)
668         {
669             if (remove(cstrFileName));
670             {
671                 CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
672                 msgError << "Error deleting backup file. errno = [" << errno << "]";
673                 fSuccess = false;
674             }
675         }
676     }
677
678     return fSuccess;
679 }
680
681 bool CWndMonitorISP::exchangeFiles() [Claim 1b, 2a, 4a, 5a, 6b, 6c, 7b, 13a, 13b]
682 {
683     CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
684     msgError << "Error occurred in exchangeFiles():";
685     char * pszLastOperation;
686     CFTPConnection * pconnFtp = NULL;
687
688     bool fSuccess = false;
689
690     try
691     {
692         //////////////////////////////////////
693         // ftp connect to the host
694         //////////////////////////////////////
695         pszLastOperation = "CInternetSession():";
696         CInternetSession sessionInternet(AfxGetAppName());
697
698         pszLastOperation = "GetFtpConnection:";
699         pconnFtp = sessionInternet.GetFtpConnection(
700             m_registry.m_strFtpAddress.c_str(),
701             m_registry.m_strFtpLogon.c_str(),
702             m_registry.m_strFtpPassword.c_str());
703
704         if (pconnFtp == NULL)
705             throw E_FAIL;
706
707         pszLastOperation = "SetCurrentDirectory():";
708         if (!pconnFtp->SetCurrentDirectory(m_registry.m_strHostDirectory.c_str()))
709             throw E_FAIL;
710
711         //////////////////////////////////////

```

```

712 // get files peculiar to this kiosk
713 ///////////////////////////////////////////////////////////////////
714 CFileNameKiosk fnKiosk;
715 fnKiosk.set("direction", "D");
716 fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
717 fnKiosk.set("date", "");
718 fnKiosk.setExtension("zip");
719 string strSearchFileName;
720 fnKiosk.getFullName(strSearchFileName);
721 pszLastOperation = "pullFtpFiles(KioskId.zip)";
722 pullFtpFiles(pconnFtp, strSearchFileName.c_str(), true);
723
724 fnKiosk.setExtension("xml");
725 fnKiosk.getFullName(strSearchFileName);
726 pszLastOperation = "pullFtpFiles(KioskId.xml)";
727 pullFtpFiles(pconnFtp, strSearchFileName.c_str(), true);
728
729 ///////////////////////////////////////////////////////////////////
730 // get application files targeted for all kiosk
731 ///////////////////////////////////////////////////////////////////
732 fnKiosk.set("kiosk_id", "KALL");
733 fnKiosk.setExtension("zip");
734 fnKiosk.getFullName(strSearchFileName);
735 pszLastOperation = "pullFtpFiles(KALL.zip)";
736 pullFtpFiles(pconnFtp, strSearchFileName.c_str(), false);
737
738 ///////////////////////////////////////////////////////////////////
739 // get lifeclinik users
740 ///////////////////////////////////////////////////////////////////
741 CFileNameDelimited fnLCUsers;
742 fnLCUsers.append("direction", "D");
743 fnLCUsers.append("file", "LCUSERS");
744 fnLCUsers.append("date", "");
745 fnLCUsers.setExtension("zip");
746 fnLCUsers.getFullName(strSearchFileName);
747 pszLastOperation = "pullFtpFiles(LCUsers)";
748 pullFtpFiles(pconnFtp, strSearchFileName.c_str(), false);
749
750 ///////////////////////////////////////////////////////////////////
751 // put daily transaction files to the host
752 ///////////////////////////////////////////////////////////////////
753 pszLastOperation = "ExportData";
754 exportData();
755
756 // set up wild card search name
757 fnKiosk.set("direction", "U");
758 fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
759 fnKiosk.set("date", "");
760 fnKiosk.setExtension("xml");
761 fnKiosk.getFullName(strSearchFileName);
762 pszLastOperation = "pushFtpFiles()";
763 pushFtpFiles(pconnFtp, strSearchFileName.c_str());
764 fSuccess = true;
765 }
766 catch(CInternetException * pE)
767 {
768     char pszErrorMessage [256];
769     pE->GetErrorMessage(pszErrorMessage, 256);
770     msgError << pszLastOperation << " Error = {0x" << std::hex << pE->m_dwError;
771     msgError << "(" << std::dec << pE->m_dwError << ")". ";
772     msgError << "Error message = [" << pszErrorMessage << "];";
773     pE->Delete();
774 }
775 catch(CException * pE)
776 {
777     char pszErrorMessage [256];

```

```
778     pE->GetErrorMessage(pszErrorMessage, 256);
779     msgError << pszLastOperation << "Error message = [" << pszErrorMessage << "];
780     pE->Delete();
781 }
782 catch(DWORD dwError)
783 {
784     msgError << pszLastOperation << "Error = [0x" << std::hex << dwError;
785     msgError << "(" << std::dec << dwError << ")] . ";
786 }
787 catch(...)
788 {
789     msgError << pszLastOperation << "Error = [Unknown Exception]";
790 }
791
792 if (pconnFtp != NULL)
793 {
794     pconnFtp->Close();
795     delete pconnFtp;
796 }
797
798 if (!fSuccess)
799     msgError.Post(_logAll);
800
801 return fSuccess;
802 }
803
804 bool CWndMonitorISP::pullFtpFiles(CFtpConnection * pconnFtp, LPCSTR pszFileSearch,
805     bool fMarkHostFile) [Claim 1b, 2a, 4a, 6c, 7b]
806 {
807     CString      cstrFileName;
808     string       strLocalFileName;
809     string       strProcessFileName;
810     CFileFind     ffLocal;
811
812     string strXferDir = m_registry.m_strLocalDirectory + "Xfer\\";
813     string strBackupDir = m_registry.m_strLocalDirectory + "Backup\\";
814     string strProcDir = m_registry.m_strLocalDirectory + "Process\\";
815
816     CFtpFileFind ffHost(pconnFtp);
817
818     // get the directory
819     BOOL fFileFound = ffHost.FindFile(pszFileSearch);
820
821     // now get each file in list
822     while (fFileFound)
823     {
824         fFileFound = ffHost.FindNextFile();
825         cstrFileName = ffHost.GetFileName();
826
827         // check to see if we've already pulled or applied file, if so, don't pull it
828         strLocalFileName = strBackupDir + (const char *) cstrFileName;
829         if (ffLocal.FindFile(strLocalFileName.c_str()))
830             continue;
831
832         strLocalFileName = strProcDir + (const char *) cstrFileName;
833         if (ffLocal.FindFile(strLocalFileName.c_str()))
834             continue;
835
836         // pull file over ftp
837         strLocalFileName = strXferDir + (const char *) cstrFileName;
838         if (!pconnFtp->GetFile(cstrFileName, strLocalFileName.c_str()))
839             throw GetLastError();
840
841         // move the file to the process directory
842         strProcessFileName = strProcDir + (const char *) cstrFileName;
843         if (rename(strLocalFileName.c_str(), strProcessFileName.c_str()))
```

```

843         throw (DWORD) errno;
844
845     if (fMarkHostFile)
846     {
847         // rename file on host, deletes do not work on cached ftp files
848         string strNewName = (const char *) cstrFileName;
849         strNewName += 'x';
850         if (!pconnFtp->Rename(cstrFileName, strNewName.c_str()))
851         {
852             DWORD dwError = GetLastError();
853             CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
854             msgError << "Unable to mark host file [";
855             msgError << (const char *) cstrFileName << "]. Error = [0x";
856             msgError << std::hex << dwError << std::dec << " (" << dwError << ") ]\n";
857             msgError.Post(_logAll);
858         }
859     }
860 }
861
862 return true;
863 }
864
865 bool CWndMonitorISP::pushFtpFiles(CFtpConnection * pconnFtp, LPCSTR pszFileSearch)
866 {
867     CString cstrFileName;
868     string strLocalFileName;
869     string strXferDir = m_registry.m_strLocalDirectory + "Xfer\\";
870     string strBackupDir = m_registry.m_strLocalDirectory + "Backup\\";
871     string strSearchFileName = strXferDir + pszFileSearch;
872
873     CFileFind ffLocal;
874     BOOL fFileFound = ffLocal.FindFile(strSearchFileName.c_str());
875     while(fFileFound)
876     {
877         fFileFound = ffLocal.FindNextFile();
878         cstrFileName = ffLocal.GetFileName();
879
880         // send the file
881         strLocalFileName = strXferDir + (const char *) cstrFileName;
882         if (!pconnFtp->PutFile(strLocalFileName.c_str(), cstrFileName))
883             throw GetLastError();
884
885         // move the file to the backup directory
886         string strBackupFileName = strBackupDir + (const char *) cstrFileName;
887         if (rename(strLocalFileName.c_str(), strBackupFileName.c_str()))
888         {
889             CLogMsgEvent msgError(LCEV_GENERIC, SVRTY_WARNING);
890             msgError << "Unable to move file [" << (const char *) cstrFileName << "] " <<
891             msgError << "to the backup directory.";
892             msgError.Post(_logAll);
893         }
894     }
895     return true;
896 }
897
898 bool CWndMonitorISP::exportData() [Claim 5a]
899 {
900     bool fSuccess = true;
901
902     string strFileName;
903     SYSTEMTIME tm;
904     GetLocalTime(&tm);
905     DATE dateNow;
906

```



```
907 // build file name
908 SystemTimeToVariantTime(&tm, &dateNow);
909 CFileNameKiosk fnKiosk;
910 fnKiosk.set("direction", "U");
911 fnKiosk.set("kiosk_id", m_registry.m_strKioskId.c_str());
912 fnKiosk.set("date", dateNow);
913 fnKiosk.setExtension("xml");
914 fnKiosk.getFullName(strFileName);
915
916 // get the exported data
917 IKCDataPtr spKCData;
918 HRESULT hr = spKCData.CreateInstance(__uuidof(KCData));
919 if (FAILED(hr))
920 {
921     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
922     msg << "Unable to instantiate IKCData object. Error = [0x";
923     msg << std::hex << hr << "].";
924     msg.Post(_logAll);
925 }
926
927 try
928 {
929     // get the exported data
930     spKCData->open();
931     _variant_t vData = spKCData->getUnexportedData();
932     string strXml = (char *) (_bstr_t) vData;
933
934     // encrypt the data
935     CEncryptor encrypt;
936     string strXmlEncrypted;
937     encrypt.Encrypt(strXml.c_str(), NULL, strXmlEncrypted);
938
939     // write the data out
940     CFile fileOut;
941     string strFileOut = m_registry.m_strLocalDirectory + "xfer\\";
942     strFileOut += strFileName;
943     fileOut.Open(strFileOut.c_str(), CFile::modeCreate | CFile::modeWrite);
944     fileOut.Write(strXmlEncrypted.c_str(), strXmlEncrypted.size());
945     fileOut.Close();
946
947     // mark all data exported
948     spKCData->markDataExported();
949
950     spKCData->close();
951
952     fSuccess = true;
953 }
954 catch(_com_error & e)
955 {
956     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
957     msg.setError(e);
958     msg.Post(_logAll);
959     fSuccess = false;
960 }
961 catch(CFileException * pEx)
962 {
963     char szError [256];
964     pEx->GetErrorMessage(szError, 256);
965     CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
966     msg << "CFile threw exception. Error = [" << szError << "].";
967     msg << "Cause = [" << pEx->m_cause << "].";
968     msg.Post(_logAll);
969     pEx->Delete();
970 }
971 catch(bool FError)
972 {
```

```
973         fSuccess = fError;
974     }
975     catch(...)
976     {
977         CLogMsgEvent msg(LCEV_GENERIC, SVRTY_WARNING);
978         msg << "Unknown exception type caught in CWndMonitorISP::exportData().";
979         msg.Post(_logAll);
980         fSuccess = false;
981     }
982     return fSuccess;
983 }
984
985 void CWndMonitorISP::buildDirectoryStructure()
986 {
987     string strBaseDir = m_registry.m_strLocalDirectory;
988     string strDir = strBaseDir + "Xfer\\";
989     mkdir(strDir.c_str());
990     strDir = strBaseDir + "Backup\\";
991     mkdir(strDir.c_str());
992     strDir = strBaseDir + "Process\\";
993     mkdir(strDir.c_str());
994     strDir += "LCUsers\\";
995     mkdir(strDir.c_str());
996     return;
997 }
998
999
```

```
1 #include "xc_OtherCommands.h"
2 #include "rs_kiosk.h"
3
4 ///////////////////////////////////////////////////////////////////
5 CXC_IMPLEMENT_FACTORY(Cxc_applyKioskTrans)
6
7 bool Cxc_applyKioskTrans::execCommand()
8 {
9     bool fSuccess = true;
10    bool fTransStarted = false;
11    m_pconn = NULL;
12    m_lAuditId = 0;
13
14    try
15    {
16        Cxs_kiosk_daily_trans rsTrans;
17
18        string strKioskId;
19        string strRecId;
20
21        getParm("kiosk_id", strKioskId);
22        getParm("rec_id", strRecId);
23
24        long lKioskId = atol(strKioskId.c_str());
25        long lRecId = atol(strRecId.c_str());
26
27        if (lKioskId == 0)
28        {
29            m_emLast << "\"kiosk_id\" is a required parameter.";
30            throw fSuccess = false;
31        }
32
33        if (lRecId == 0)
34        {
35            m_emLast << "\"rec_id\" is a required parameter.";
36            throw fSuccess = false;
37        }
38
39        m_pconn = m_pcoClient->getConnection();
40        if (m_pconn == NULL)
41        {
42            m_emLast.setError(m_pcoClient->getLastError());
43            throw fSuccess = false;
44        }
45
46        rsTrans.setActiveCommand("cmdGetTrans");
47        rsTrans.setParameter("kiosk_id", _variant_t(lKioskId));
48        rsTrans.setParameter("rec_id", _variant_t(lRecId));
49
50        if (!m_pconn->execute(rsTrans))
51        {
52            m_emLast.setError(m_pconn->getLastError());
53            throw fSuccess = false;
54        }
55
56        if (rsTrans.isEmpty())
57        {
58            m_emLast << "There are no transactions for kiosk_id [" << lKioskId << "], \r\n";
59            m_emLast << lRecId << "].";
60            throw fSuccess = false;
61        }
62
63        // get the transactions in xml format
64        string strXml;
65        rsTrans.getField("data", strXml);
```

```
66 CXmlDocument xdocTrans(strXml.c_str());
67 if (!xdocTrans.IsReady())
68 {
69     string strParseError;
70     xdocTrans.GetParserError(strParseError);
71     m_emLast << "Unable to parse XML transactions. kiosk_id = [" << lKioskId;
72     m_emLast << "], rec_id [" << lRecId << "]. Error = [" << strParseError << "\n";
73     throw fSuccess = false;
74 }
75 CXmlElement elTable;
76 fTransStarted = m_pconn->beginTrans();
77 m_lAuditId = getAuditId();
78 // apply alternate id
79 if (xdocTrans.GetItem("kc_id_map", &elTable))
80 {
81     xdocTrans.pushCurrent(&elTable);
82     Crs_kc_id_map rsAId;
83     rsAId.SetActiveCommand("applyTrans");
84     fSuccess = applyTransactions(xdocTrans, rsAId);
85     xdocTrans.popCurrent();
86 }
87 // apply kiosk users
88 if (fSuccess && xdocTrans.GetItem("kc_user", &elTable))
89 {
90     xdocTrans.pushCurrent(&elTable);
91     Crs_kc_user rsUser;
92     rsUser.SetActiveCommand("applyTrans");
93     fSuccess = applyTransactions(xdocTrans, rsUser);
94     xdocTrans.popCurrent();
95 }
96 // apply blood pressures
97 if (fSuccess && xdocTrans.GetItem("kc_blood_pressure", &elTable))
98 {
99     xdocTrans.pushCurrent(&elTable);
100     Crs_kc_blood_pressure rsBP;
101     rsBP.SetActiveCommand("applyTrans");
102     fSuccess = applyTransactions(xdocTrans, rsBP);
103     xdocTrans.popCurrent();
104 }
105 // apply weights
106 if (fSuccess && xdocTrans.GetItem("kc_weight", &elTable))
107 {
108     xdocTrans.pushCurrent(&elTable);
109     Crs_kc_weight rsWeights;
110     rsWeights.SetActiveCommand("applyTrans");
111     fSuccess = applyTransactions(xdocTrans, rsWeights);
112     xdocTrans.popCurrent();
113 }
114 fTransStarted = false;
115 string strStatus;
116 string strReason;
117 if (fSuccess)
118 {
119     m_pconn->commitTrans();
120     strStatus = "p";
121 }
122
123
124
125
126
127
128
129
130
```

```
131     else
132     {
133         m_pconn->rollbackTrans();
134         strStatus = 'E';
135         m_emLast.getError(strReason);
136     }
137
138     // mark transaction processed
139     rsTrans.setActiveCommand("cmdUptKioskTrans");
140     rsTrans.setParameter("kiosk_id", _variant_t(1KioskId));
141     rsTrans.setParameter("rec_id", _variant_t(1RecId));
142     rsTrans.setParameter("status", _variant_t(strStatus.c_str()));
143     rsTrans.setParameter("reason", _variant_t(strReason.c_str()));
144     rsTrans.setParameter("audit_id", _variant_t(1L));
145     if (!m_pconn->execute(rsTrans))
146     {
147         m_emLast.setError(m_pconn->getLastError());
148         throw fSuccess = false;
149     }
150 }
151 catch(_com_error & e)
152 {
153     m_emLast.setError(e);
154     fSuccess = false;
155 }
156 catch(bool fBool)
157 {
158     fBool;
159     fSuccess = false;
160 }
161 catch(...)
162 {
163     m_emLast.setError("Unkown exception raised. [Command:applyKioskTrans]");
164     fSuccess = false;
165 }
166
167 if (fTransStarted)
168 {
169     if (fSuccess)
170         m_pconn->commitTrans();
171     else
172         m_pconn->rollbackTrans();
173 }
174
175 return fSuccess;
176 }
177
178 bool Cxc_applyKioskTrans::applyTransactions(CXmlDocument &xdocData, CSdoRecordset & rsCmdRecSet)
179 {
180     bool fSuccess = false;
181     string strTag;
182     m_emLast.clear();
183
184     try
185     {
186         //Active command is expected to be set prior calling.
187
188         //get the Sdo command pointer
189         CSdoCommand * pcmdSdo = rsCmdRecSet.getActiveCommand();
190
191         //get the table element from the stack
192         CXmlElement elTable;
193         xdocData.getCurrent(&elTable);
194         string strAttrValue;
195         elTable.getAttribute("t", strAttrValue);
```

```
196     if (strAttrValue[0] != 't')
197     {
198         m_emLast.clear();
199         m_emLast << "Expecting XML table element to have attribute t=\"t\".";
200         throw fSuccess = false;
201     }
202
203     CXMLElement elRow;
204     bool fRows = elTable.getFirst(&elRow); //get the "row" Element
205
206     while (fRows) //process for all "row" Elements
207     {
208         //get "row"tag
209         elRow.getTag(strTag);
210         if (strcmp(strTag.c_str(), "row") != 0)
211         {
212             m_emLast.clear();
213             m_emLast << "Expecting XML element \"row\". Found \"" << strTag << "\"";
214             fSuccess = false;
215             break;
216         }
217
218         //Set all the parameters
219         xdocData.pushCurrent(&elRow);
220
221         pcmdSdo->clearParms();
222         if (pcmdSdo->setParms(xdocData) == false)
223         {
224             string strError;
225             pcmdSdo->getLastError(strError);
226             m_emLast.setError(strError.c_str());
227             fSuccess = false;
228             break;
229         }
230
231         if (m_lAuditId)
232             pcmdSdo->setParm("audit_id", _variant_t(m_lAuditId));
233
234         xdocData.popCurrent();
235
236         // update the table
237         if (!(fSuccess = m_pconn->execute(rsCmdRecSet)))
238         {
239             m_emLast.setError(m_pconn->getLastError());
240             fSuccess = false;
241             break;
242         }
243
244         //fetch next row to update.
245         fRows = elTable.getNext(&elRow);
246     }
247
248     fSuccess = true;
249 }
250 catch(_com_error & e)
251 {
252     m_emLast.setError(e);
253     fSuccess = false;
254 }
255 catch(bool fBool)
256 {
257     fBool;
258     fSuccess = false;
259 }
260 catch(...)
```

```
261 {  
262     m_emLast.setError("Unkown exception raised. [CxcLCBrokerModify::execute  
    (parameters)]");  
263     fSuccess = false;  
264 }  
265  
266 return fSuccess;  
267 ;  
268
```

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☒ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**